1978

Optimization of Query Evaluation Algorithms

S. B. Yao

Report Number:
78-283

OPTIMIZATION OF QUERY EVALUATION ALGORITHMS*

S.B. Yao**
Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907

TR 283

ABSTRACT

A model of database storage and access is presented. The model represents many evaluation algorithms as special cases, and helps to break a complex algorithm into simple access operations. Generalized access cost equations associated with the model are developed and analyzed. Optimization of these cost equations yields an optimal access algorithm which can be synthesized by a query subsystem whose design is based on the modular access operations.

Key Words and Phrases: relational data model, query language, data manipulation language, query optimization, data base optimization, inverted file.

---

* This research was supported by the National Science Foundation under grant number MCS 76-16604.

** Presently on leave at New York University, Department of Computer Applications and Information Systems.

1. Introduction

While regular batch processing and reporting remain the main functions of database systems, there is an increasing demand to respond quickly to simple data access requests. The use of non-procedural query languages greatly simplifies data access by non-programming (or ad hoc) users. Query languages available in existing systems usually allow users to express what is to be accessed without specifying how it is to be accessed. Users are deliberately made aware of only the data definitions, not the storage definitions of the system, for the purpose of providing a more "friend" system-user interface.
As a result, the burden of specifying access strategies must be assumed by the query subsystem of a database system.

The efficiency of access strategies directly affects the performance of the query subsystem. Although many query subsystems were implemented, only a few that employ the relational data model gave some considerations to the performance of its access algorithms: Astrahan and Chamberlin (1975) considered the optimization of the "restriction" operation. Gotlieb (1975) analyzed the computation of the "join" operation in isolation. Pecherer (1975) studied the evaluation of relational operators in an abstract machine. Wong and Youssefi (1976) developed heuristics to decompose a complex query into a sequence of simple queries. Rothnie (1975) designed methods to eliminate unnecessary accesses using "feedback" information. However, most of these authors did not consider secondary storage access , which is usually the dominating component of the response time for a query. The access costs are estimated from simple cost models without considering the storage structures of the data base system. While Blasgen and Eswaran (1977) do compare secondary storage accesses for several access strategies, their cost model is still simplistic and there is no reason to believe that the cases studied are in any sense complete.

The freedom in selecting access strategies by the query subsystem offers an opportunity for the system to optimize its performance. The query optimization must satisfy two criteria: effectiveness and completeness. The optimization is effective if its cost is less than the benefit, which is the access cost saving resulting from the optimization. It is complete if the optimal (or near optimal) strategy produced by the optimization procedures can be realized by the system.

Heuristics may be used to improve upon a given access strategy, such as those developed by Smith and Chang (1975). Alternatively, a set of access strategies may be implemented in the query subsystem. For a given query, the system computes the cost equations corresponding to the implemented strategies, and selects the one with minimal cost (Astrahan 1976). The first approach usually employs gross heuristics operations developed from intuitive observations. As there does not exist a set of universally applicable heuristics, this may exclude from consideration valid optimal strategies, or can even sometimes make queries less efficient. Hence the effectiveness and the overall improvement of the query subsystem efficiency are difficult to justify. The second approach suffers from the fact that only a limited number of access strategies can be implemented in practice. It is, therefore, obviously incomplete.

The approach used in this paper is based on the observation that most of the queries expressed by non-programmer users are simple (Yao 1978a)and that a complex query can be decomposed into a set of simple queries (Wong and Youssefi 1976). We studied in detail the optimization of a restricted class of "two-variable" simple queries. It is discovered that query evaluation algorithms can be constructed using a small set of access operations which form the basis of a query evaluation model.

The generality of the model enables the systematic analysis of a very large

collection of access strategies which previously had to be analyzed individually. The associated cost model which takes into consideration detailed data base storage structures is used to compute the cost of access strategies in terms of secondary storage accesses. An optimizer for the determination of the optimal access strategy by minimizing the cost model is developed. Using the access operations of the model, an adaptive query subsystem can also be developed which generates optimal access strategies according to the instructions of the optimizer. Thus, the present approach is shown to be both effective and complete for simple queries.

It should be pointed out that our approach requires the relatively infrequent complex queries to be decomposed by a heuristic procedure before optimization, and the result may be sub-optimal. Before the modeling is extended to include complex queries, the possibility of using other approaches for highly complex queries should not be ruled out. Further analysis and experimentation are required to identify an appropriate approach for complex queries.

The basic concepts used to develop the query evaluation model are derived from works relevant to the relational data model. However, since the storage structure requirements of a relational system are similar to those of other types of systems, the applications of the models developed in this paper are not limited to the relational systems.

In the next section, we define the basic concepts for the data and storage structures considered. Section 3 defines the type of simply query considered and illustrates the decomposition of a complex query into simple queries. Section 4 presents a graphic representation for the query evaluation algorithm. Section 5 presents a generalized model for the evaluation of two-variable queries. The associated cost equations for the model are derived in Section 6. Finally, Section 7 compares the cost estimations of the model with previous results and describes the design of a self-optimizing query subsystem as a sample application of the query evaluation model.

2. Data and Storage Models.

The definitions and notations used to describe the data model follow those given in Astrahan (1976), Codd (1970) and Yao (1977). Although some of the definitions appear previously in the literature, they are included here for completeness. We start by considering a set of entities for which data are to be recorded.

Definition 2.1. An attribute is a binary relation between the entity set and a value set. The set of values which participate in the attribute is called the active domain of the attribute. Each element in the active domain is called a value.

Definition 2.2. A file F defined over a set of attributes $A_F=\{a_1,\ldots,a_n\}$ is a set of records, each consisting of values from the active domains of the attributes. Any smallest set of attributes whose values uniquely identify records in the file is called the primary key of the file.

Files in a database may be conveniently viewed as a collection of tables; each row of the table corresponds to a record and each column of the table corresponds to an active domain. The value of the attribute a of the record r is denoted r(a).

Definition 2.3. Two attributes are compatible if their active domains are defined over the same value set.

Definition 2.4. Let a and b be compatible attributes from files F and G, respectively. The relationship S(a,b) between a and b is a set of ordered pairs $\{(r_1, r_2) \mid r_1 \in F, r_2 \in G \text{ and } r_1(a) = r_2(b)\}$. S is one-to-many (1:M), denoted $a \rightarrow b$, if for any $r_2 \in G$ there exists at most one $r_1 \in F$ such that $(r_1, r_2) \in S$. The relationship is one-to-one (1:1), denoted $a \leftrightarrow b$, if $a \rightarrow b$ and $b \rightarrow a$. Otherwise the relationship is many-to-many (M:N), denoted $a \nleftrightarrow b$.

Relationships between attributes of files are implicit, as their existence is implied by data values. They are important in data accessing and cross-referencing. The performance of these activities may be improved by designing additional access paths and organizations.

Let p(r) denote the address (pointer) of the record r, and P(r) denote the set of pointers stored with the record r. The following access paths for relationships are defined:

Definition 2.5. A link is a storage realization of a one-to-many relationship $a \rightarrow b$. The parent link of $a \rightarrow b$ is defined if $p(r_1) \in P(r_2)$ for any $(r_1,r_2) \in S(a,b)$. The child link of $a \rightarrow b$ is defined if $p(r_2) \in P(r_1)$ for any $(r_1,r_2) \in S(a,b)$. The chain link of $a \rightarrow b$ is defined if for any r_i, the set $\{r_j | (r_i,r_j) \in S(a,b)\}$ is ordered into a sequence $r_{i1}, r_{i2}, \ldots, r_{in_i}$ such that $p(r_{i1}) \in P(r_i)$ and $p(r_{i,j+1}) \in P(r_{ij})$ for all $j=1,\ldots,n_i-1$.

In other words, the parent link provides a parent pointer in each child; the child link stores all children pointers in parents as "pointer arrays"; and the chain link organizes the parent and children into a linked list.

Definition 2.6. The clustering links between the file F and the files G_k, $k=1,\ldots,n$ are links defined over the relationships $a \rightarrow b_k$, $a \in A_F$, $b_k \in A_{G_k}$, $k=1,\ldots,n$ such that for any record $r_i \in F$ the sets of records $G_k' = \{r_j \mid r_j \in G_k, (r_i,r_j) \in S(a,b_k)\}$, $k=1,\ldots,n$ are stored sequentially following the record r_i. The file F is called the parent file of the children files G_k, $k=1,\ldots,n$.

The definition is recursive in that it is possible for a child file to become a parent file having its own children files. To make the clustering links well-defined, we do not permit a file to become a child file of more than one parent file. Thus the clustering links define a tree structure on files. The storage sequence of records is determined by the hierarchical order of traversal (or preorder traversal) of the tree (IMS, Schkolnick 1977).

Definition 2.7. An index on the attribute a is a binary relation $I_a = \{(v,p(r)) \mid v = r(a), r \in F\}$. An indexing (index search) for a given value v_j of the attribute a yields the sequence $P_{v_j} = \{p_i \mid (v_j,p_i) \in I_a\}$ $i=1,\ldots,n_j$, such that $p_k < p_\ell$ for all $k<\ell$. An index scan on the index I_a yields the sequence $\{P_{v_j}\}_{j=1,\ldots,n}$, $v_k < v_\ell$ for all $k<\ell$.

That is, indexing a value produces pointers to records containing this value. An index scan produces all pointers in the index, major-sorted on values and minor-sorted on pointers. Since indices permit direct access of records in the file, they are stored in structures that provide rapid indexing and index scan.

Definition 2.8. A clustering index is an index such that if $p_k \in P_{v_k}$, $p_\ell \in P_{v_\ell}$ and $v_k < v_\ell$ then $p_k < p_\ell$. In other words, the pointers in $(P_{v_j})_{j=1,\ldots,n}$ define a total ordering.

It is assumed that the records are stored in pages (or blocks). A clustering index causes the records in a file to be stored in a sorted order on the values of the index. Thus it is possible to retrieve all the records in a file by accessing each page at most once, using pointers obtained by an index scan.

Definition 2.9. A file F is a hashing file if addresses of records in F are determined by a key-to-address transformation $T:K \to A$ and an overflow function $O: A \to A$ such that $p(r) = O(T(r(k)))$ where $r \in F$, $r(k)$ is the primary key of r, K is the set of all primary keys of F, and A is the set of all addresses.

Definition 2.10. Given m files $F_1,\ldots,F_m$, the database storage state at time t is a triple $S_m(t) = (\mathcal{C},\mathcal{D},\mathcal{X})$, defined as follows: $\mathcal{C} = (\mathcal{C}_0,\mathcal{C}_1,\mathcal{C}_2,\mathcal{C}_3)$ is the link state where

$$\mathcal{C}_0 = [c_{ij}], i,j=1,\ldots,m;\ c_{ij} = \begin{cases} 1 \text{ if there is a clustering link from } F_i \text{ to } F_j \\ 0 \text{ otherwise} \end{cases}$$

$$\mathcal{C}_1 = [c'_{ij}], i,j=1,\ldots,m;\ c'_{ij} = \begin{cases} 1 \text{ if there is a parent link from } F_i \text{ to } F_j \\ 0 \text{ otherwise} \end{cases}$$

$$\mathcal{C}_2 = [c''_{ij}], i,j=1,\ldots,m;\ c''_{ij} = \begin{cases} 1 \text{ if there is a child link from } F_i \text{ to } F_j \\ 0 \text{ otherwise} \end{cases}$$

$$\mathcal{C}_3 = [c'''_{ij}], i,j=1,\ldots,m;\ c'''_{ij} = \begin{cases} 1 \text{ if there is a chain link from } F_i \text{ to } F_j \\ 0 \text{ otherwise} \end{cases}$$

$\vartheta = (\vartheta_1, \ldots, \vartheta_m)$ is the index state where

$\vartheta_i = [c_j], j=1,\ldots,n_i$ is the index state of F_i such that

$$c_j = \begin{cases} -1 & \text{if there is no index for the attribute } a_j \\ 0 & \text{if there is a non-clustering index for the attribute } a_j \\ 1 & \text{if there is a clustering index for the attribute } a_j \end{cases}$$

$\chi = [h_i], i=1,\ldots,m$ is the hashing state where

$$h_i = \begin{cases} 1 & \text{if } F_i \text{ is a hashing file} \\ 0 & \text{otherwise} \end{cases}$$

The parameters defined by the data base storage state describe the physical storage structure of a data base system. They are useful in determining the applicability of access algorithms and in estimating the access cost of algorithms.

3. Query Type

It is often desirable to access data stored in a data base system using a non-procedural query language. A high-level query expressed in a query language does not contain explicit link information or access strategies; it only specifies the condition for the response set of the target records. However, in order to specify access requests which involve more than one file, the implicit relationship must be utilized. Query languages are developed for the relational data model, since this model does not provide access-oriented link information. Various types of query language are also available in many existing database systems. This section defines the type of query analyzed in this paper.

Definition 3.1. A clause is a triple of the form $(a\ \theta\ v)$ or $(a\ \theta\ b)$, where a and b are attributes, v is a value of the attribute a, and θ is one of the symbols from the set $\{=, \neq, <, >, \leq, \geq\}$. A predicate P is a conjunctive normal form of clauses.

Denote A_D as the set of attributes in the predicate D and $\mathcal{F}_D$ as the set of files containing the attributes in A_D. The notion of an n-variable query is defined as follows.

Definition 3.2. An (n-variable)* query is a triple $Q_n = (D,\mathcal{F}_D,A)$ where D is a predicate, $\mathcal{F}_D = \{F_1,\ldots,F_n\}$ is a set of files containing attributes in D, and A is a target list of attributes whose values are to be accessed.

Definition 3.3. A query graph for a query $Q_n = (D,\mathcal{F}_D,A)$ is a graph $(\mathcal{F}_D,S)$ where $\mathcal{F}_D$ is the set of nodes and $S = \{(F_i,F_j) \mid a \in A_{F_i},\ b \in A_{F_j},\ i \neq j,\ (a\ \theta\ b) \in D\}$ is the set of arcs.

The query graph provides a concise representation for queries. It is convenient to label the arcs (F_i,F_j) by $(a\ \theta\ b) \in D$ where $a \in A_{F_i}$ and $b \in A_{F_j}$. The nodes F_i are labelled by the sub-predicate D_i that contains only the attributes in A_{F_i} and by the target attributes in $A_i = \{a \mid a \in A \text{ and } a \in A_{F_i}\}$.

* Here the "variable" is defined in the sense of relational calculus (Codd 1971). We assume that one variable is defined for each file.

Example 3.1. Given the following files:

PROJECT (P#, Title, Location, D#)

DEPARTMENT (D#, Budget)

EMPLOYEE (E#, Sex, D#)

EQUIPMENT (Eq#, Type, D#)

A query statement "List all projects located in LAFAYETTE that have FEMALE employees and which are managed by a department which owns equipment-type TRUCK" may be represented as a three-variable query $Q_3 = (D, \mathcal{F}_D, A)$ where

D = (PROJECT.Location = 'LAFAYETTE') $\wedge$ (PROJECT.D# = EMPLOYEE.D#) $\wedge$ (EMPLOYEE.Sex = 'FEMALE') $\wedge$ (PROJECT.D# = EQUIPMENT.D#) $\wedge$ (EQUIPMENT.Type = 'TRUCK')

$\mathcal{F}_D$ = {PROJECT, EMPLOYEE, EQUIPMENT}

A = {PROJECT.P#, PROJECT.Title}

The query graph of Q_3 is shown in Figure 3.1.

Definition 3.4. Given two records $r_1 = <v_1, \ldots, v_m>$ and $r_2 = <u_1, \ldots, u_n>$, the concatenation of r_1 with r_2 is the record defined by: $r_1 \cdot r_2 = <v_1, \ldots, v_m, u_1, \ldots, u_n>$. The Cartesian product of two files F and G is defined as $F \boxtimes G = \{ r_1 \cdot r_2 \mid r_1 \in F, r_2 \in G \}$.

Let $\boxtimes \mathcal{F}_D = F_1 \boxtimes \ldots \boxtimes F_n$ denote the Cartesian product of the files $F_1, \ldots F_n \in \mathcal{F}_D$. The semantics of the query Q_n is defined by specifying its response set.

Definition 3.5. The response set of the query $Q_n = (D, \mathcal{F}_D, A)$ is the file $R_{Q_n} = \{s = <u_1, \ldots, u_m> \mid u_i = r(a_i)$, $a_i \in A$, $r \in \boxtimes \mathcal{F}_D$ and r satisfies D$\}$.

The definition of query response set offers an approach for query evaluation, namely: first compute the Cartesian product $\boxtimes \mathcal{F}_D$, then examine each record in $\boxtimes \mathcal{F}_D$ for satisfying D, and construct records in the response set. This procedure is not efficient since the size of $\boxtimes \mathcal{F}_D$ could be enormous. The complexity of a high-order query (i.e. large n) makes it difficult to design an efficient evaluation algorithm. In fact, it was shown that the optimal evaluation of queries is a very hard (NP) problem (Bernstein 1978). The optimization of query evaluation can only be effective for low-order queries.

It was shown that any n-variable query may be reduced to two-variable queries by decompositions and substitutions (Wong 1976). For example, the reduction of Q_3

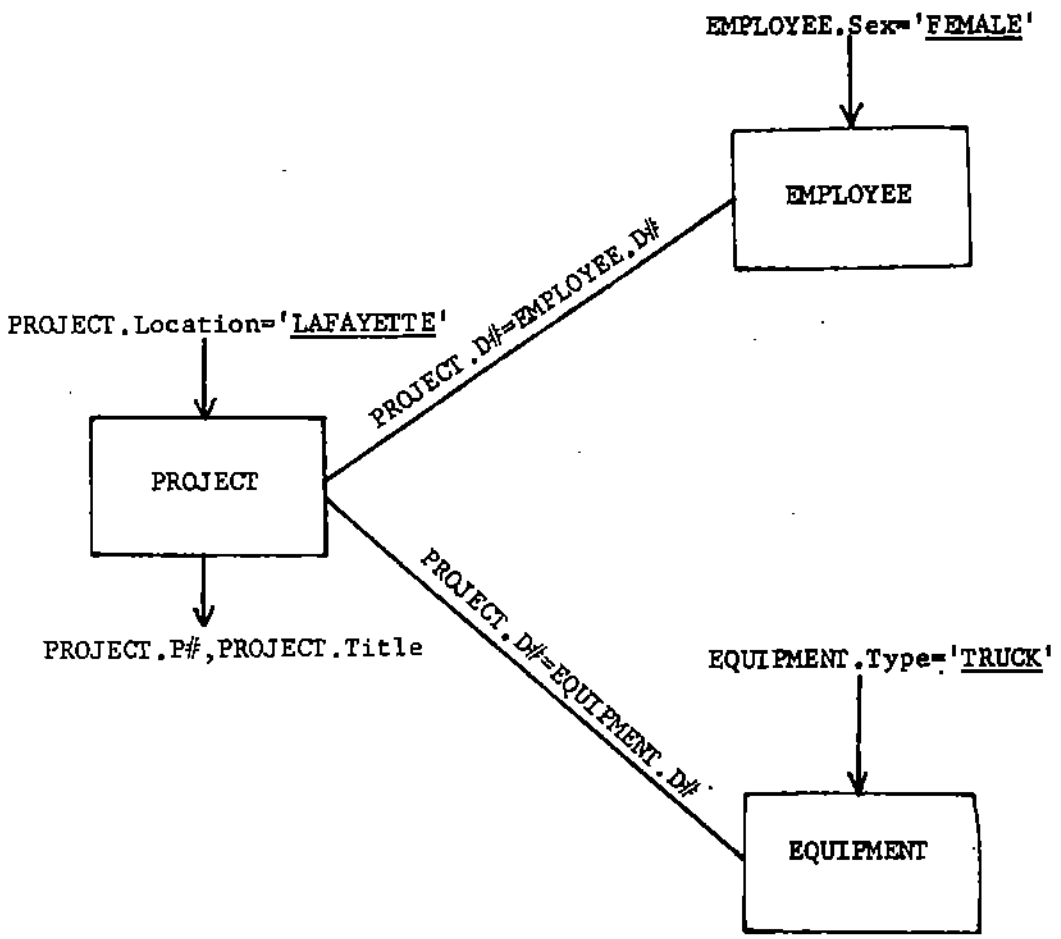


Figure 3.1 The Query Graph of Q_3 in Example 3.1.

in Example 3.1 is illustrated in Figure 3.2. Heuristics were also designed to guide the reduction process (Wong 1976). In this paper, we will show that the query complexity of two-variable queries is sufficiently reduced so that the precise effects of storage structure on query evaluation can be analyzed.

In order to analyze the evaluation of two-variable queries, it is necessary to define some elementary operations that were first introduced by Codd (1971).

Definition 3.6. A restriction R is a mapping $R(D,F) = R_R$ where D is a predicate, F is a file, and R_R is the response set defined as $R_R = \{r \mid r \in F$ and r satisfies D $\}$.

The restriction operation solves an important type of one-variable query. More precisely, if $Q_1 = (D,\{F\},A_F)$ then $R_{Q_1} = R(D,F)$.

Definition 3.7. A join J of two files F and G is defined as a mapping $J((a\theta b),F,G) = R_J$ where a and b are compatible attributes of the files F and G, respectively, and R_J is the response set containing record concatenations of two files on "matching" values of attributes a and b. That is, $R_J = \{f \cdot g \mid f \in F, g \in G, f(a)\ \theta\ g(b)\}$. The attributes a and b are called join attributes.

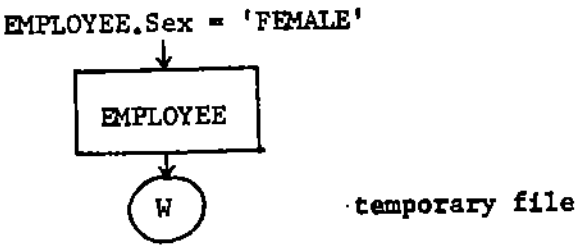
The join operation solves an important type of two-variable query. More precisely, if $Q_2 = (D,\{F,G\},A)$ where $D = (a\ \theta\ b)$, $a \in A_F$, $b \in A_G$, and $A = A_F \cup A_G$, then $R_{Q_2} = J((a\theta b),F,G)$.

Definition 3.8. A projection P is a mapping $P(F,A) = R_P$ where F is a file, $A \subseteq A_F$, and R_P is the response set which is the subset of the file F with all values of attributes not in A deleted. That is, $R_P = \{s = \langle u_1,\ldots,u_m\rangle \mid u_i = r(a_i)$, $a_i \in A$ and $r \in F\}$.

The projection operation also solves an important type of one-variable query. Obviously, if $Q_1 = ('TRUE',\{F\},A)$ where $a \in A_F$, then $R_{Q_1} = P(F,A)$.

The operations defined above may be combined to solve a more general type of query. For example, a general one-variable query $Q_1 = (D,\{F\},A)$ where $A \subseteq A_F$ is solved by $R_{Q_1} = P(R(D,F),A)$. The general type of two-variable query is of the form $Q_2 = (D,\{F,G\},A)$ where $A \subseteq A_F \cup A_G$. In this paper, we are especially

1. Decomposition. Sub-Query Q1:



2. Substitution. For each value d_i of the attribute EMPLOYEE.D# in W, repeat sub-query Q2:

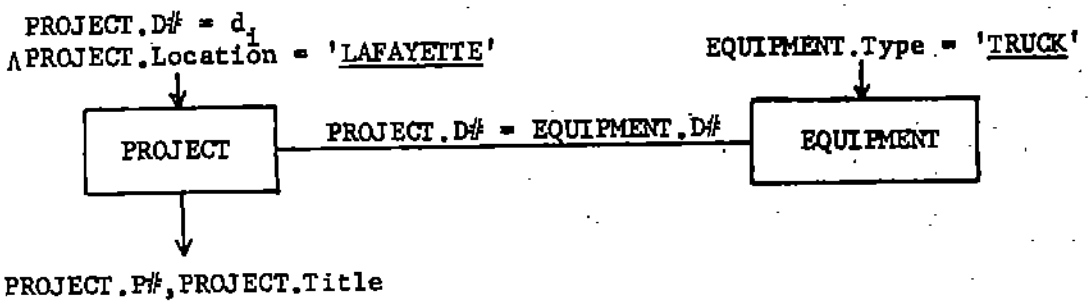


Figure 3.2 Reduction of the Query Q3 in Example 3.1

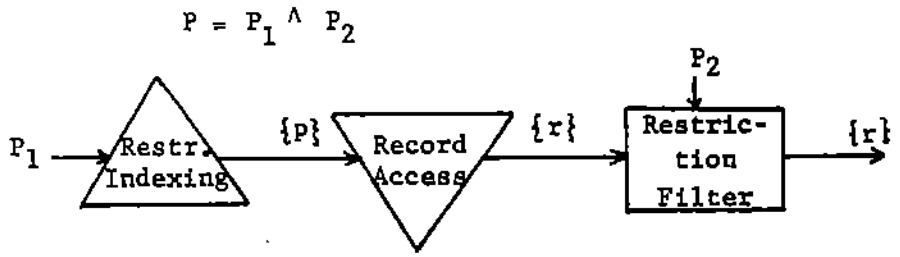


Figure 4.1 One-Variable Query Evaluations

interested in the case where $D = D_1 \wedge D_2 \wedge D_3$, $A_{D_1} \subseteq A_F$, $A_{D_2} \subseteq A_G$, $D_3 = (a \theta b)$, $a \in A_F$ and $b \in A_G$. We denote this type of two-variable query as $Q_2' = (D, \{F, G\}, A)$. The query type Q_2' is solved by the operations $(P(J((a \theta b), R(D_1, F), R(D_2, G)), A)$.

4. Query Evaluation

We have seen that various methods for evaluating queries are available. The methods differ in the way they use indices and links, in the order of various operations performed, and in their particular applicability. The collection of existing methods is not exhaustive, since new methods may be created from small variations of existing methods. In this section and the following one, evaluation of one-variable and simple two-variable queries is examined. A query evaluation model is then introduced for the systematic synthesis and comparison of evaluation algorithms for general two-variable queries.

The evaluation of one-variable queries has been previously investigated (Astrahan 1975, Yao 1977). Let $A = \{a_i \mid i = 1, \ldots, n\}$ be the set of attributes in the predicate P. An attribute a is said to be indexed if there exists an index for values of a. Therefore, the set A can be partitioned into two mutually exclusive subsets:

$A_1 = \{ a \in A,$ a is indexed$\}$

and $A_2 = \{ a \in A,$ a is not indexed $\}$.

The evaluation of one-variable queries is illustrated graphically in Figure 4.1. The predicate in the one-variable query is decomposed into the form $P_1 \wedge P_2$, where P_1 contains only the indexed attributes, and P_2 contains only the non-indexed attributes. Figure 4.1 shows that P_1 is resolved by accessing the indices; this results in a set of pointers $\{P\}$. Records $\{r\}$ located at pointed locations are accessed and examined for satisfying the

predicate P_2. Obviously, if none of the attributes in the predicate P are indexed (i.e. P_1 is TRUE), then the step Restriction-Indexing may be eliminated; and if all the attributes in the predicate are indexed (i.e. P_2 is TRUE), then the Restriction-Filtering step is unnecessary. It is also possible to define decompositions such that P_1 contains a subset of the indexed attributes. The "optimal" decomposition is investigated by Astrahan (1975) An extension is discussed in Appendix A.

The evaluation of a join is illustrated in Figure 4.2b. Input to the i-th join filter is a file containing records, grouped (and sorted) by the join values. This is obtained by first accessing the records, and then sorting them on the join attribute, as shown in Figure 4.2a. Let $\{ v_i, \{r_i\} \}$ denote the set containing records grouped by their join values. The join filters intersect the two input files, and produce subfiles $\{(v_i, \{r_i\}) \mid v_i \in V_1 \cap V_2\}$, $i = 1,2$, where V_i is the set of join values of the file i. Finally, records in the two subfiles with matching join values are concatenated on the join value.

Figure 4.2c shows that it is possible to join, instead of records, the record pointers, and access records at a later stage. This requires, however, the existence of indices on the join attributes. The input pointer sets (grouped and sorted by the join values) are obtained by accessing these indices.

The evaluation of a general two-variable query can now be examined. First, consider the following examples.

Example 4.1. Given two files F and G, define a two-variable query of the form $Q_2 = (D, \mathcal{F}_D, S)$

where $D = P \wedge Q \wedge (a \theta b)$

$\mathcal{F}_D = \{F, G\}$

$S = A \cup B$, $A \subset A_F$ and $B \subset A_G$

$A_P \subset A_F$, $A_Q \subset A_G$, $a \in A_F$ and $b \in A_G$

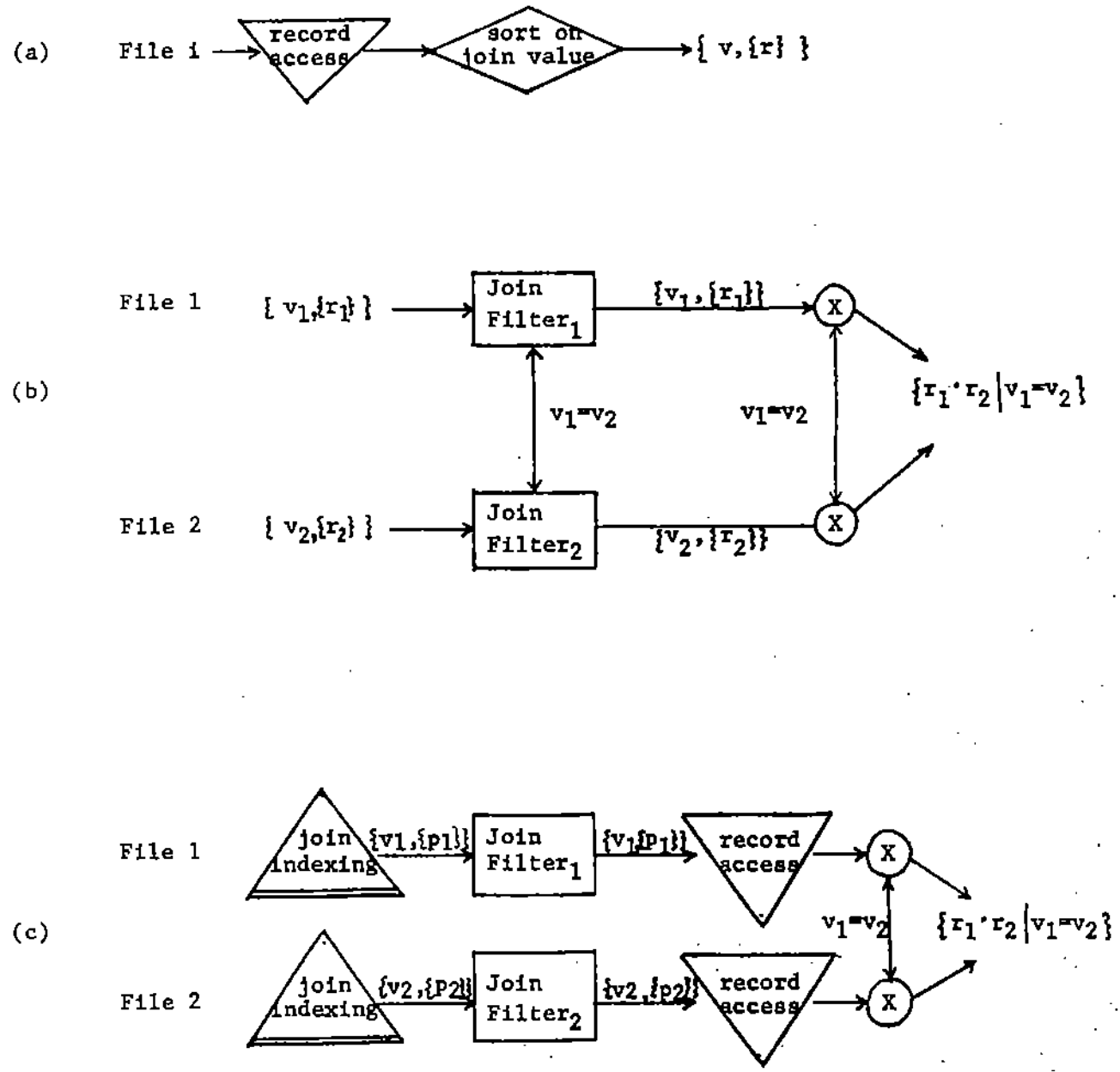


Figure 4.2 Simple Two-Variable Query Evaluation

This query is graphically represented in Figure 4.3.

Assuming indices exist for both files, the query in Example 4.1 may be evaluated by two well-known algorithms illustrated in the following examples.

Example 4.2. (Sort Files). Scan the files F and G using restriction indices, and create two temporary files T_1 and T_2. T_1 and T_2 contain records which satisfy the restrictions, and consist of attributes that are either the projection attributes or the join attributes. Sort T_1 and T_2 on the join attributes. Scan the sorted files and perform the join.

Example 4.3. (TID Algorithm). Use restriction indices on F and G to obtain pointers for records which satisfy the restrictions. Store the sorted pointers in temporary files T_1 and T_2. Scan the join indices of F and G for join participation; intersect with T_1 and T_2 respectively. Retrieve records with the resulting pointers and perform join and projection.

It is convenient to describe the algorithms using the graphic notation developed. A graphic representation for Example 4.2 is shown in Figure 4.4a. The first three operations on each file correspond to the one-variable sub-queries R(P,F) and R(Q,G). The projection operations $P(F,A \cup \{a\})$ and $P(F,B \cup \{b\})$ performed next are modified to include the join attributes. The rest of the algorithm corresponds to the two-variable sub-query $J((a\theta b),F,G)$, except that the original projections P(F,A) and P(G,B) are performed as the final steps.

The graphic description for the "TID algorithm" is similarly given in Figure 4.4b. The subqueries are mixed in evaluation. The join index operation produces record pointers grouped by ascending join values. The join filter eliminates record pointers that do not participate in both input sets, and produces grouped pointers $\{v,\{p\}\}$. These pointers are intersected with those obtained from the restriction operation, resulting in pointers $\{v,\{p\}\}$ to records that satisfy both the restriction index and the join. These records are retrieved by the access operation, projected, and checked for satisfying the complete restriction. Finally, the concatenations of records are produced.

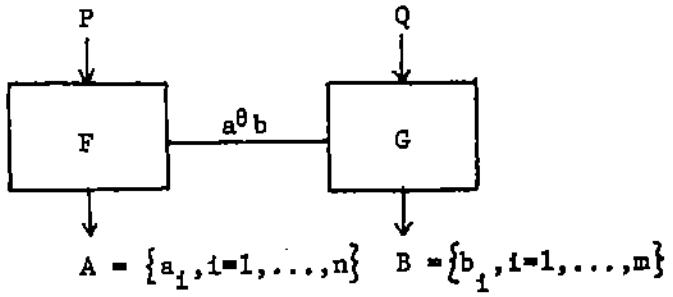
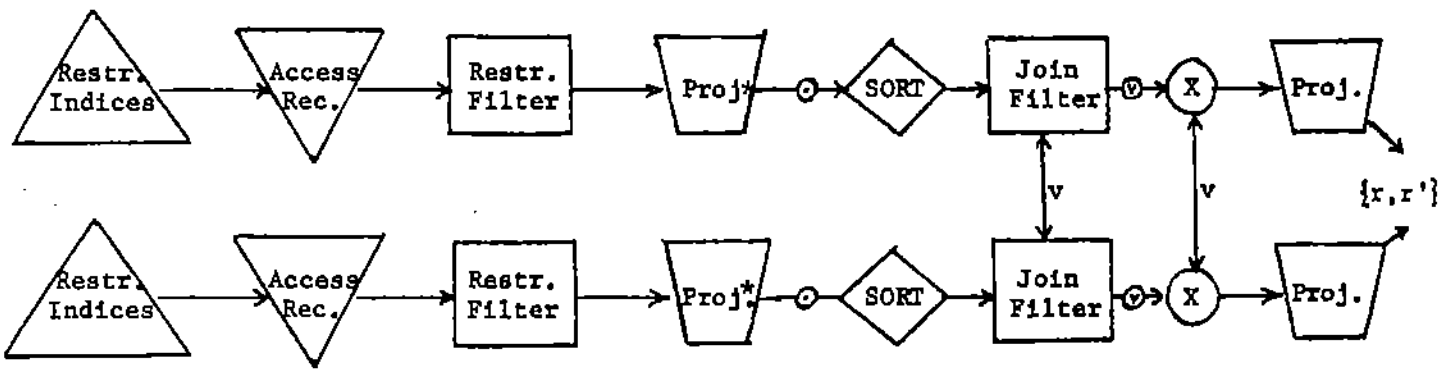
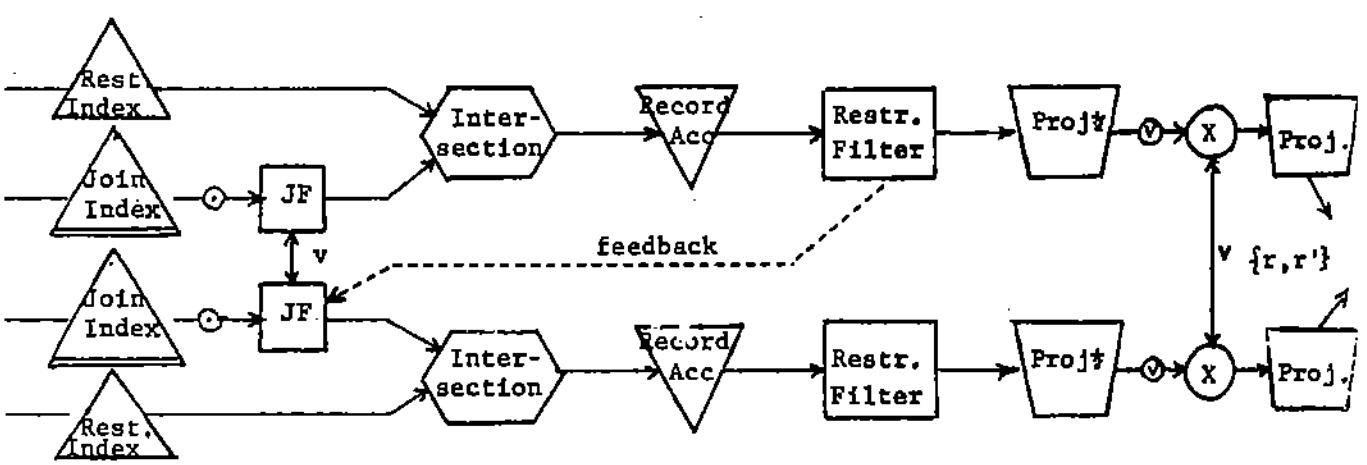


Figure 4.3 Graphic Representation of a Two-Variable Query

a. Sort Files



b. TID Algorithm



* Proj. is a partial projection which includes the join attribute.

Figure 4.4 Graphic Representation of Query Evaluation

Rothnie discovered that the pointers or records passing the join filter can be further filtered by the use of the "feedback" information (Rothnie 1975). Consider an instance of the output of the two join filters $v_1,\{p\}$ and $v_2,\{p\}$ where $v_1 = v_2$. If the records corresponding to $v_1,\{p\}$ are later rejected by the restriction filter, then the pairs $v_2\{p\}$ should also be rejected. Assuming $v_1,\{p\}$ is processed before $v_2,\{p\}$ by the algorithm, the feedback sends a message from the restriction filter on File 1 to the join filter on File 2 notifying the latter to reject $v_2,\{p\}$. In effect, this reduces the amount of data to be processed for File 2.

The operations in these graphs may be viewed as asynchronous processes with particular input and output specifications. They are not independent, however, since the input of one operation may be the output of another For operations that can function on partial inputs, varying degrees of "pipelining" are possible. To see this, consider that the access operation can retrieve record(s) for one record pointer, for a subset of record pointers, or for the complete set of input record pointers. In order to control the degree of pipelining, two flow notations are used in the graphs:

(1) Complete-hold ⊙ - The complete output of the previous operation must be obtained before the next operation can start.

(2) Join-value-hold ⊛ - All output of the previous operation pertinent to a particular join value must be obtained before the next operation can start.

Note that other types of holds are possible, and that the selection of an "optimal" degree of pipelining is an open question. For simplicity, only two types of holds are considered here. In Figure 4.4a, the use of complete-holds in front of the sort operations reflects the implicit

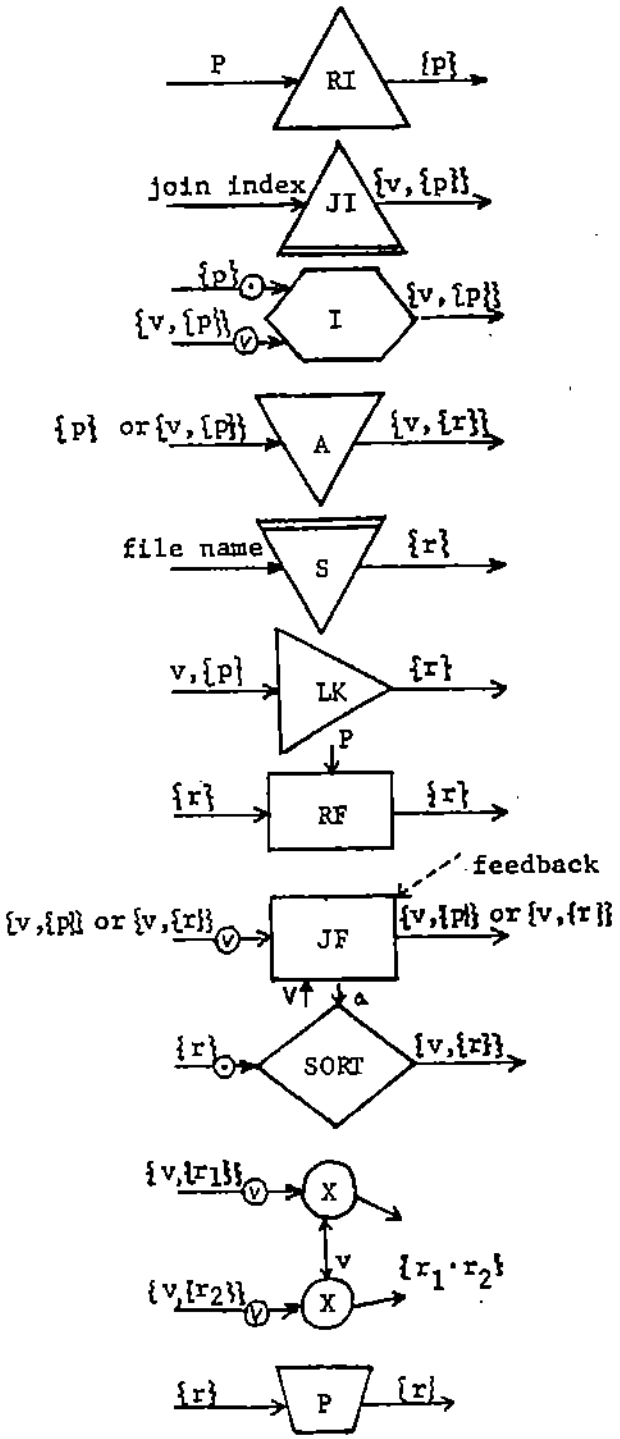
assumption that the insertion sort is not employed. The use of join-value-holds before the join filters is necessary, since the record groups are processed for one join value at a time. The pipelining degree is arbitrary where unspecified.

The two examples in Figure 4.4 introduced a total of nine types of operations. These operations plus a sequential file-scan operation and a link scan operation are summarized in Fig. 4.5. We will show in the next section that it is possible to synthesize most of the query evaluation algorithms using a model based on only these operations as components.

5. Query Evaluation Model

Inspection of query evaluation algorithms reveals that there are four basic tasks performed: restriction (R), join (J), record access (A), and projection (P). Different query evaluation algorithms correspond to different sequences and methods of executing these tasks on the two files involved. For example, it is clear from Figure 4.4a that the "Sort Files" algorithm corresponds to the sequences RAPJ/RAPJ for the two files, and the "TID Algorithm" corresponds to the sequences RJAP/RJAP.

For each file there are 4! = 24 possible arrangements of the four operations. We note from the examples that the operations for the two files are independent except for the interactions between the join filters and between the concatenation operations where they must be "synchronized" for each join value. Since different sequences may be used for the two files, consideration of the sequencing along would yield a total of 4! x 4! = 576 distinct query evaluation algorithms! This is an over-estimation, however, if we note that the projection operations cannot be performed until the records are accessed. Still, the number of distinct sequences is at least 12x12=144. The query evaluation model developed in this section first considers the operations on one file and divides



Restriction Indexing
Input: restriction predicate
Output: set of pointers to records satisfying the predicate

Join Indexing
Input: join index
Output: record pointers grouped by join values

Intersection
Input: two sets of pointers, one of them grouped by join values
Output: intersection of the two input sets, grouped by join values

Record Access
Input: set of pointers, may be grouped by join values
Output: records grouped by join values

Sequential Scan
Input: file name
Output: records in the file

Link Scan
Input: pointers to records containing a particular join value
Output: all records containing the join value

Restriction Filter
Input: restriction predicate and a set of records
Output: records satisfying the predicate

Join Filter
Input: pointers (or records) grouped by join values and the join values processed by the other JF
Output: pointers (or records) grouped by join values which also appear in the other JF

Sorting
Input: records and the sorting (join) attribute
Output: records grouped by ascending join values

Concatenation
Input: two sets of records grouped by join values
Output: concatenation of records with matching join values

Projection
Input: set of records
Output: records containing only the projected attributes (duplicates are removed)

Figure 4.5 Summary of Search Operations

the algorithms into classes according to the sequences created by the three operations R, J, and A. Variations within each class are then considered. Finally, the combined algorithms for two files are analyzed.

Class 1 (RAJ). As can be seen in Figure 5.1, the three major operations are performed in the order of R, A and J. The evaluation sequences is a concatenation of a restriction sub-query (Figure 4.1) and a join sub-query (Figure 4.2a,b). The projection operation may be inserted at any step after the record access A. However, if the projection is inserted before the join filter, then it must include the join attribute and an additional projection after the join is required. The location of the projection determines three versions in this class. We note that the Sort Files algorithm in Example 4.2 corresponds to exactly the third version in this class.

Class 2 (JAR). By interchanging the operations J and R, a new class of access algorithms is defined. A variation of the join query is used (Figure 4.2c) and the restriction filter is inserted after the record access. Similarly to the previous class, projection may be inserted at two locations. This introduces two versions in this class. Furthermore, concatenation may be performed before the records are accessed, to produce pairs of pointers instead of pairs of records. This defines the third version of this class of algorithms.

Class 3 (JRA&RJA). Algorithms in this class perform the access operations last. Since the join and restriction using indices may be done independently, their order is immaterial. The results of restriction indexing and join filter are combined using an intersection operation which produces pointers to those records that satisfy both the restriction and the join. Similar to the previous case, the projection may be inserted at two locations, and the concatenation may be relocated either to follow the join or to follow the intersection operation. These variations define a total of four versions in this class. We note that the TID Algorithm in Example 4.3 corres-

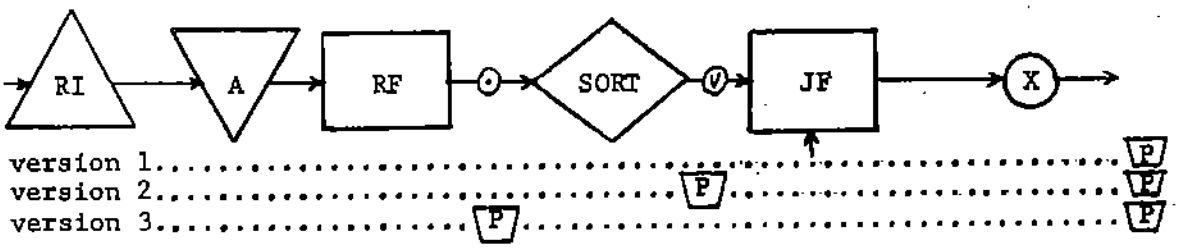


Figure 5.1 Graphic Representation of Class 1 Algorithms (RAJ)

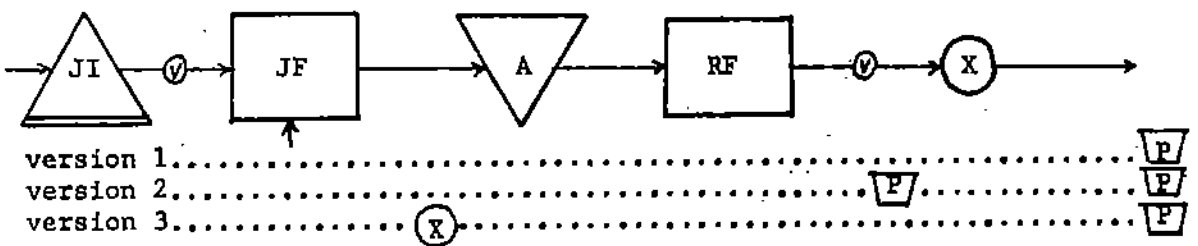


Figure 5.2 Graphic Representation of Class 2 Algorithms (JAR)

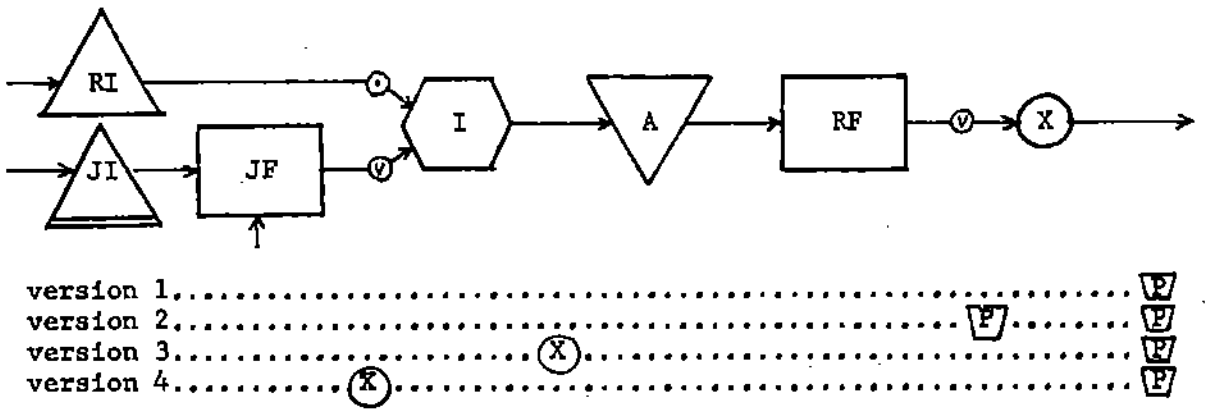


Figure 5.3 Graphic Representation of Class 3 Algorithms (JRA & RJA)

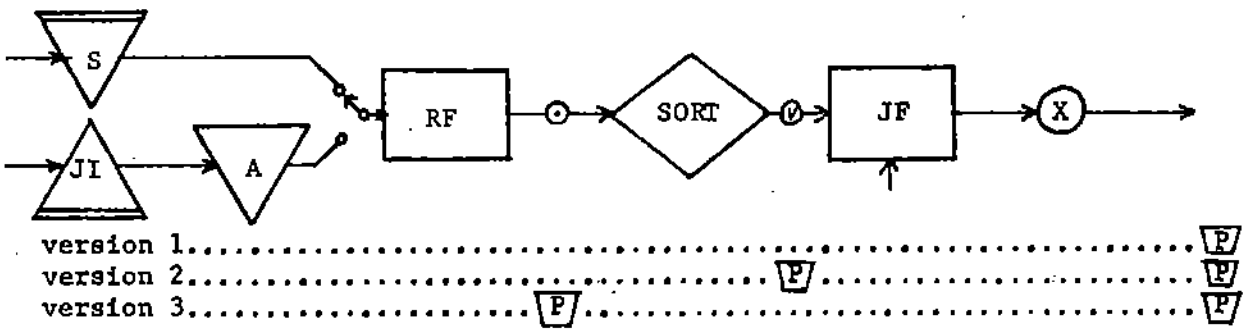


Figure 5.4 Graphic Representation of Class 4 Algorithms (ARJ)

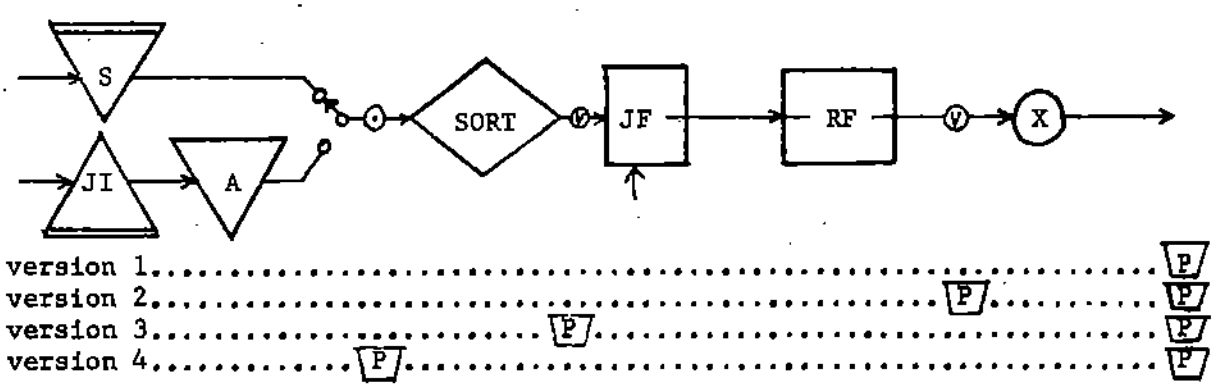


Figure 5.5 Graphic Representation of Class 5 Algorithms (AJR)

ponds precisely to the second version of this class.

Class 4 (ARJ). It is sometimes desirable to first access records before performing any restriction or join operations. This is especially the case where there exist no indices on the restriction and the join attributes. In order to retrieve the records, the file must be scanned using some existing access paths. One obvious method which suggests itself is to scan the file sequentially. Alternatively, if the records in the file are distributed in a large storage space, then it may be preferable to first obtain record pointers by serially processing an existing index. These two alternatives are shown in Figure 5.4. The records retrieved are examined for satisfying the restriction, and a join sub-query follows. The reader may note the similarity between this class and the class 1 algorithm. Comparing Figures 5.1 and 5.4, we see the two classes have identical "back end" operations. This is expected, since they both perform the join sub-query as the final step. There are also three versions in this class.

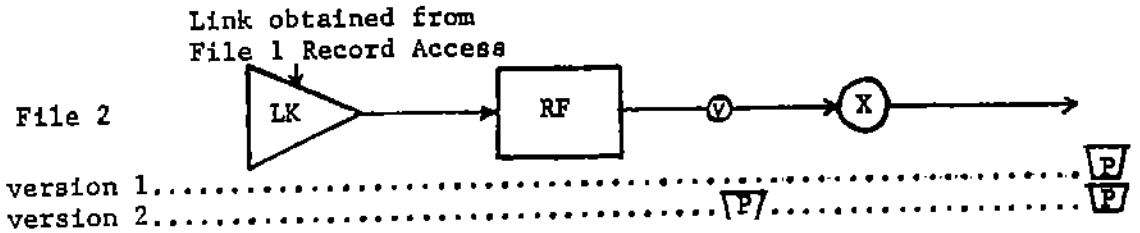
Class 5 (AJR). Reversing the join and restriction order in the previous class results in the relocation of the restriction operation. Figure 5.5 shows that the restriction filter is now performed after the join filter. Other parts of the algorithm remain unchanged. In addition to the three possible versions in the previous class, this class has a fourth version that performs the projection just after the restriction filter.

Class 6 (LAR). This class of algorithm applies to file G when an existing link from file F to file G defined over the join attributes is used. Recall that the links are stored as pointers in records of file F pointing to the joined records in file G, regardless of the cardinality of the link (i.e. link to parent or link to children). To obtain the link pointers, records in file F must be accessed; that is, the algorithm used for file F must be

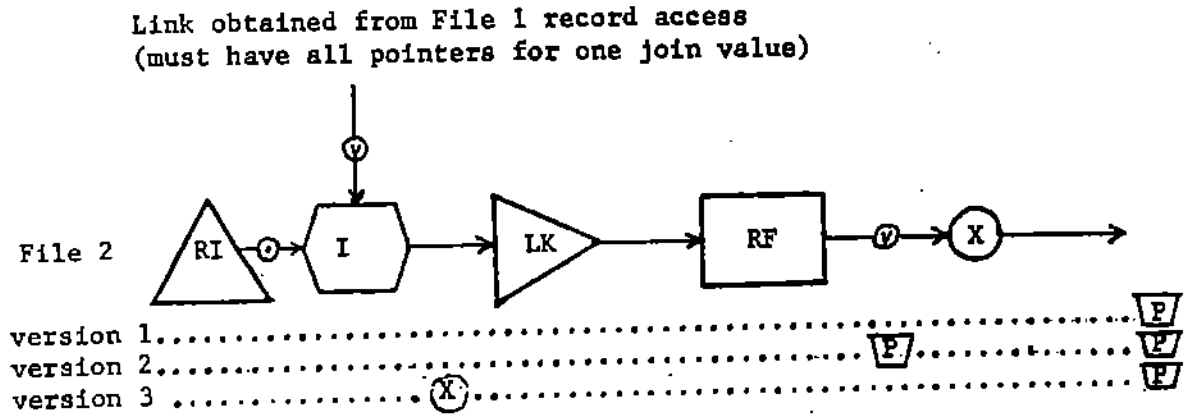
Class 1, 4 or 5 (Figure 5.6a). Using the link pointers, records in file G are accessed, examined for satisfying the restriction, and concatenated with records in file F. Again, note that the only difference between this class and Class 2 is in the way the join is performed. Link pointers may be implemented as a linked list which stores in records of file F only the pointer for the first child record in file G. The location of the projection defines two versions of this class. It is not possible to relocate the concatenation as in Class 2, since not all points are available prior to the link access operation. We note that this class includes the conventional "find parent" and "find children" methods.

Class 7 (LRA&RLA). The link traversal method can also be used to replace the join operation in Class 3. The result is very similar to the previous class, except that the pointers obtained from the link traversal are further "filtered" by the intersection operation before the records are accessed. All the pointers to records satisfying the join must be presented before the intersection operation. This means that all the link pointers must be stored in records of file F using a "multi-child" method such as the children link. The three possible variations of this class follow directly from Class 3. Version 4 of Class 3 does not apply because the records in File 1 are already "concatenated" with link points.

The above classification of algorithms includes the six possible permutations of the three basic operations. Three additional sequences are created by substituting the join with the link-traversal. It is not possible to substitute a link traversal for the joins in Classes 4 and 5, since the link traversal performs the join without accessing the records in file G.



(a) Direct Link Access (Class 6, LAR)



(b) Filtered Link Access (Class 7, LRA and RLA)

Figure 5.6 Graphic Representation of Class 6 and 7 Algorithms

We now define the rules to combine the algorithms for two files. Let i denote the algorithm class i and i/j denote the type of algorithm combining classes i and j. For example, the type 1/6 indicates the application of class 1 (RAJ) on the file F and the application of class 6 (LAR) on the file G after traversing the link from F to G. The combination rule can be stated as follows:

A:= 1|2|3|4|5

B:= 1|4|5

L:= 6|7

TYPE:= A/A|B/L

This rule gives 25+6 = 31 basic types of algorithms. The representation can be refined to include versions within each class. In what follows, we will use ik/jℓ to indicate the application of class i version k on F and class j version ℓ on G. With this refinement, it is easy to see that the model can generate at least 17*17+10*5 = 339 distinct types of algorithms.

Given a particular data base state, not all types of algorithm are applicable, since each class of algorithm has different requirements. Let $a_1,,,,,a_r$ denote the restriction attributes, and a_i denote the join attribute. Using the data base storage state parameters defined earlier, Table 5.1 shows the storage structure requirement for each algorithm class.

	Class						
	1	2	3	4	5	6	7
Restr. Index $C_1+C_2+\ldots+C_r>-r$	✓		✓				✓
Join Index $C_i \geq 0$		✓	✓				
Link Type $C_{ij}+C'_{ij}+C''_{ij}+C'''_{ij}>0$						✓	✓
Link Type $C'_{ij}+C''_{ij}>0$							✓

Table 5.1 Requirements of Algorithm Classes

6. Cost Model

In order to compute the cost of the access algorithms, we first define the cost of each search operation. The cost is computed for a particular state of the storage model. The parameters that describe the state of the storage model are given in Section 2. It is assumed that the storage is organized into blocks or pages, and that the cost for the access algorithms is measured in terms of page accesses.

It is further assumed that the restriction predicate C is expressed in the Conjunctive Normal Form and that some of the attributes in the predicate are indexed in the storage model. It was shown by Astrahan (1976) that only a subset of these indices can be profitably accessed in order to reduce the number of record accesses. The algorithm for selecting indices for accessing is given in Appendix A. Define the access predicate $D = (d_{11} \vee \ldots \vee d_{1n_1}) \wedge \ldots \wedge (d_{m1} \vee \ldots \vee d_{mn_m})$ to be a subpredicate of C such that D contains only the indexed attributes selected and each d_{jk} is a disjunction containing clauses of an indexed attribute.

Figure 6.1 summarizes the parameters used in the cost model. The cost equations for each access operations are stated below. The detailed derivation is given in Appendix B.

Restriction Indexing:
$$R(r_i) = \sum_{j=1}^{m} \sum_{k=1}^{n_j} u_{jk} \left(\log_z r_i + \frac{r_i s_{jk}}{b} \right)$$

Join Indexing:
$$I(r_i) = \frac{r_i}{b}$$

Record Access:
$$A(\alpha, \beta) = x + (p_i - x) \frac{\alpha r_i p_i}{p_i}$$

where $x = p_i (1 - 1/p_i)^{r_i \beta}$,

α is the selectivity of the clustering index, and

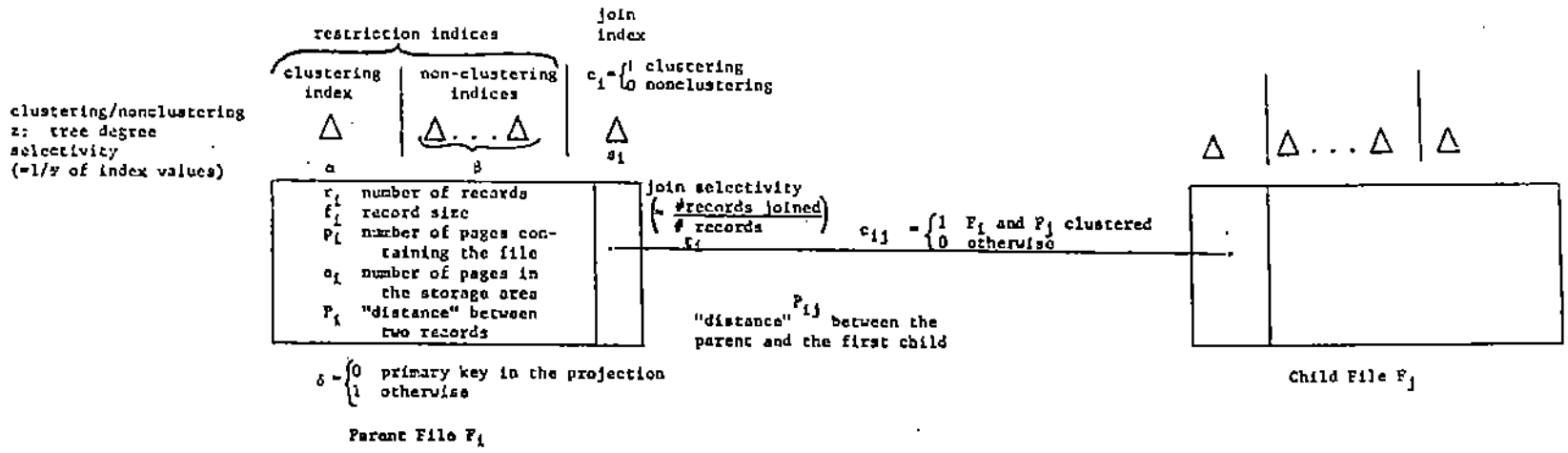
β is the selectivity of the nonclustering indices.

c_{jk} predicate clause

d_{jk} access predicate clause

u_{jk} number of values accessed

s_{jk} index selectivity



b: page size

Figure 6.1 Summary of Parameters in the Cost Model

Sequential Scan: $S(e_i)=e_i$

Sorting*: $T(r,f_i)=2\frac{r\cdot f_i}{b}\log_n\frac{r\cdot f_i}{b}$

Joining: $J(r,f_i)=\frac{r\cdot f_i}{b}$

Concatenation: $C(r,f_i)=\frac{r\cdot f_i}{b}$

Concatenation (of pointers): $C_1(r)=\frac{r^2 s_i t_i}{b}$

Projection: $P_o(r)=T(r,f_i)+\frac{\delta}{s_i}T(rs_i,f_i)$

Projection (after SORT): $P(r)=\frac{1}{s_i}T(rs_i,f_i)$

File size after projection: $H(r)=MIN(r;\prod_{a_j\in A}\frac{1}{s_j})$

Link Access (1:m): $L(\alpha',\beta')=c_{ij}\beta'(P_{ji}+(\alpha'-1)P_i)$

$+(1-c_{ij})\beta'(c_i(1+(\alpha'-1)P_i)+(1-c_i)\alpha')$

where $\alpha'=r_i t_i$

and $\beta'=r_j t_j$

Link Access (m:1): $L'(\alpha')=c_{ij}\alpha' P_{ij}+(1-c_{ij})\alpha'$

We note that the hashing is treated as a non-clustering index in the record access operation, except that it does not incur any cost in the restriction indexing operation.

Once the cost of each access operation is obtained, the cost of the access algorithms for one file can be derived by combining the individual costs. Let q denote the selectivity of the restriction predicate. The access predicate is resolved by the restriction indexing and has a less restrictive selectivity: $q=\gamma+\xi$ where γ is the selectivity of the clustering index, and ξ is the selectivity of the non-clustering indices. It follows that the restriction filter after indexing has the selectivity $p=\frac{\hat{q}}{q}$. The detailed derivation of q, $\hat{q}$, γ, and ξ are given in Appendix C.

---

* r refers to the size of the file being accessed.

We next analyze the effect of feedback assuming the records in File 1 are processed before those in File 2. Recall that the records in File 2 having a particular join value can be eliminated, if all the records in File 1 passing the join filter with the same join value are later rejected by the restriction filter. Since the selectivity of the File 1 restriction filter is $p = \hat{q}/q$ (p is called the feedback parameter), the probability for a File 1 record to be rejected is 1-p. Since there are $r_i s_i$ records in File 1 having the same join value, the probability for a join value to be rejected is $(1-p)^{r_i s_i}$. Therefore, the probability for a File 2 record to survive the feedback is given by $F(p) = 1-(1-p)^{r_i s_i}$. The feedback is possible only when File 1 uses algorithm class 2, 3 and 5, where the join filter is used before a restriction filter. In other cases there is no feedback and we set p=1.

We now compute the access costs for the algorithm classes applied on one file. In most cases, this is simply to sum up the costs of the access operations involved. Let $C_{\alpha\beta}$ denote the access cost of algorithm class α version β. Also define

$$\rho = \begin{cases} 0 \text{ if the algorithm is applied on File 1} \\ 1 \text{ if the algorithm is applied on File 2.} \end{cases}$$

1) Cost of Class 1 (RJA)

As can be seen from Figure 5.1, the only access components that have associated costs are RI, A, SORT, J and C. Summing up these costs we have

$$C_{11} = R(r_i)+A(Y,\xi)+T(r_i\hat{q},f_i)+J(r_i\hat{q},f_i) + F(p)^{\rho}C(r_i\hat{q}t_i,f_i).$$

Note that the term $F(n)^{\rho}$ accounts for the feedback effect when C_{11} is used on File 2 (i.e. ρ=1). When the projection is performed before the join, it reduces the size of the file to be joined. The record size is also reduced from f_i to g_i. The cost is given by:

$$C_{12} = R(r_i)+A(Y,\xi)+T(r_i\hat{q},f_i)+P(r_i\hat{q})+J(H(r_i\hat{q}),g_i)+ F(p)^{\rho}C(H(r_i\hat{q}t_i),g_i).$$

When the projection is performed before the sort operation, a complete sorting is required to remove the duplicate records. However, the cost of the original

sort is reduced.

$$C_{13} = R(r_i)+A(\gamma,\xi)+P_o(r_i\hat{q})+T(H(r_i\hat{q}),g_i)+J(H(r_i\hat{q}),g_i) + F(\rho)^{\rho}C(H(r_i\hat{q}t_i),g_i).$$

2) Cost of Class 2 (JAR)

As shown in Figure 5.2, the record pointers obtained by the join indexing are further filtered by the join. Assume that the join attribute selectivity is t_i. The parameters for the record access cost are defined as:

clustering index selectivity $\alpha = \begin{cases} t_i & \text{if the join attribute has a clustering index} \\ 0 & \text{otherwise.} \end{cases}$

nonclustering index selectivity $\beta = t_i - \alpha$

The cost of the algorithm is therefore:

$$C_{21} = I(r_i)+J(r_i,1) +F(p)^{\rho}\left[A(\alpha,\beta)+C(r_i\hat{q}t_i,f_i)\right].$$

We note that $J(r_i,1)$ represents the cost of joining pointers. If the projection is performed before the concatenation, we have

$$C_{22} = I(r_i)+J(r_i,1) + F(p)^{\rho}\left[A(\alpha,\beta)+P(r_i\hat{q})+C(H(r_i\hat{q}t_i),g_i)\right].$$

If the concatenation operation is performed on pointers before records are accessed, the cost of the access algorithm becomes

$$C_{23} = I(r_i)+J(r_i,1)+ F(p)^{\rho}\left[C_1(r_i)+A(\alpha,\beta)\right].$$

The feedback parameter defined by this class (i.e. when this class is used on File 1) is p=q.

3) Cost of Class JRA/RJA

As shown in Figure 5.3, this class of algorithms attempts to resolve both the join and the restriction using indices before records are accessed. The record access cost depends also on the effect of the clustering join index. We define the following parameters:

$$\gamma' = \begin{cases} t_i & \text{if the join attribute has a clustering index} \\ \gamma & \text{otherwise} \end{cases}$$

$$\xi' = \begin{cases} \xi & \text{if the join attribute has a clustering index} \\ \xi t_i & \text{otherwise} \end{cases}$$

$$q' = r' + \zeta'$$

The cost of the algorithm is given by

$$C_{31} = r(r_i)+I(r_i)+J(r_i,1) + F(p)^{\rho}[A(\gamma',\xi')+C(r_i\hat{q}t_i,f_i)].$$

If the projection is done before the concatenation, we have

$$C_{32} = R(r_i)+I(r_i)+J(r_i,1) + F(p)^{\rho}[A(\gamma',\xi')+P(r_i\hat{q}t_i)+C(H(r_i\hat{q}t_i),g_i)]$$

If the concatenation is performed on record pointers satifying both the restriction and the join indices, we have the cost

$$C_{33} = R(r_i)+I(r_i)+J(r_i,1) + F(p)^{\rho}[C_1(r_iq')+A(\gamma',\xi')]$$

By moving the concatenation beyond the intersection operation, we have

$$C_{34} = R(r_i)+I(r_i)+J(r_i,1) + F(p)^{\rho}[C_1(r_i)+A(\gamma',\xi')]$$

It is clear that $C_{34} \geq C_{33}$ since $C_1(r_i) \geq C_1(r_iq')$. Therefore C_{34} is eliminated from further consideration. When this class of algorithm is used on File 1, the restriction indices have a selectivity q and the restriction filter has a selectivity q'/q, which defines the feedback parameter.

4) Cost of Class ARJ

The cost is obtained by examining Figure 5.4. The first step is to select the best method for sequentially retrieving records from the file. The sequential access via index is similar to join indexing, except that in this case the index is not a join index, and the selectivity is 1. By defining

$$\alpha' = \begin{cases} 1 \text{ if the access index is a clustering index} \\ 0 \text{ otherwise} \end{cases}$$

$$\beta' = 1 - \alpha'$$

we have

$$C_{41} = MIN(S(e_i);I(r_i)+A(\alpha',\beta'))+T(r_i\hat{q},f_i)+J(r_i\hat{q},f_i)\ F(p)^{f}\ C(r_i\hat{q}t_i,f_i).$$

If the projection is performed before the join filter, we have

$$C_{42} = MIN(S(e_i);I(r_i)+A(\alpha',\beta'))+T(r_i\hat{q},f_i)+P(r_i\hat{q})+J(H(r_i\hat{q}),g_i)\ F(p)^{f}C(H(r_i\hat{q}t_i),g_i).$$

If the projection is further moved beyond the sort, we have

$$C_{43} = MIN(S(e_i);I(r_i)+A(\alpha',\beta'))+P_o(r_i\hat{q})+T(H(r_i\hat{q}),g_i)+J(H(r_i\hat{q}),g_i)+F(p)^{r}C(H(r_i\hat{q}t_i),g_i$$

5) Cost of Class AJR

The cost of this class is obtained similarly to the class ARJ (see Figure 5.5). The performance of this class is slightly inferior, however, since it does not use the restriction filter to immediately reduce the file size. The cost equations are

$$C_{51} = MIN(S(e_i);I(r_i)+A(\alpha',\beta'))+T(r_i,f_i)+J(r_i,f_i)+F(p)^{\rho}C(r_iqt_i,f_i)$$

$$C_{52} = MIN(S(e_i);I(r_i)+A(\alpha',\beta'))+T(r_i,f_i)+J(r_i,f_i)+F(p)^{\rho}[P(r_i\hat{q}t_i)+C(H(r_iqt_i),g_i)]$$

$$C_{53} = MIN(S(e_i);I(r_i)+A(\alpha',\beta'))+T(r_i,f_i)+P(r_i)+J(H(r_i),g_i)+F(p)^{\rho}C(H(r_i\hat{q}t_i),g_i)$$

$$C_{54} = MIN(S(e_i);I(r_i)+A(\alpha',\beta'))+P_o(r_i)+T(H(r_i),g_i)+J(H(r_i),g_i)+C(H(r_i\hat{q}t_i),g_i).$$

Comparing each term in the equation with that of the class AJR, we found $C_{51} > C_{41}$, $C_{52} > C_{42}$, $C_{53} > C_{43}$ and $C_{54} > C_{43}$. Since the applicability of the two classes is identical, the class AJR is eliminated from further consideration. The feedback parameter defined by this class is $p = \hat{q}$.

6) Cost of Class LAR

From Figure 5.6, the cost is simply the sum of the link access cost and the concatenation cost. The first two cases are for the (1:m) link access:

$$C_{61} = L(r_it_i,r_jt_j)+C(r_i\hat{q}t_i,f_i).$$

If the projection is performed before concatenation, we have

$$C_{62} + L(r_it_i,r_jt_j)+P(r_i\hat{q}t_i)+C(H(r_i\hat{q}t_i),g_i)$$

Two similar cases are defined for (m:1) link access:

$$C_{63} = L'(r_it_i)+C(r_i\hat{q}t_i,f_i)$$

$$C_{64} = L'(r_it_i)+P(r_i\hat{q}t_i)+C(H(r_i\hat{q}t_i),g_i)$$

7) Cost of Class LRA/RLA

Figure 5.7 shows that the restriction indices are used to reduce the link access. Similar to the class LAR, the cost equations for the four cases are

$$C_{71} = R(r_i)+L(r_iqt_i,r_jt_j)+C(r_i\hat{q}t_i,f_i)$$

$$C_{72} = R(r_i)+L(r_iqt_i,r_jt_j)+P(r_i\hat{q}t_i)+C(H(r_i\hat{q}t_i),g_i)$$

$$C_{73} = R(r_i)+L'(r_iqt_i,r_jt_j)+C(r_i\hat{q}t_i,f_i)$$

$$C_{74} = R(r_i)+L'(r_iqt_i,r_jt_j)+P(r_i\hat{q}t_i)+C(H(r_iqt_i),g_i).$$

There are two additional cases defined for relocation of the concatenation operation. The (1:m) link access case is

$$C_{75} = R(r_i)+C_1(r_iqt_i) + L(r_iqt_i,r_jt_j).$$

The (m:1) link access case is

$$C_{76} = R(r_i)+C_1(r_iqt_i) + L'(r_iqt_i,r_jt_j).$$

Using the cost equations for one file developed above, the complete query cost equation can be obtained. The cost for the algorithm class $ik/j\ell$ is simply

$$C_{ik/j\ell} = C_{ik}^{p=0} + C_{j\ell}^{f=1}$$

By close examination of the cost equations, the properties of $C_{ik/j\ell}$ may be investigated. However, these equations are simple enough to lend themselves to efficient evaluation. In the next section, evaluations of $C_{ik/j\ell}$ are compared with some known special cases. A system which adapts to the optimal $C_{ik/j\ell}$ is also described.

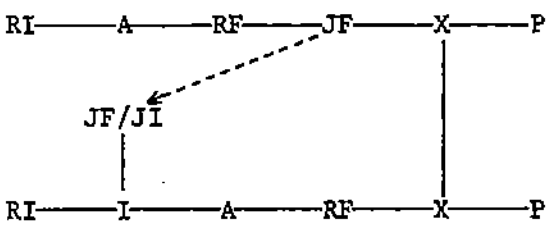
7. EVALUATION AND OPTIMIZATION

One of the applications of the cost model developed in this paper is to evaluate and compare different access algorithms. In what follows, we first compare a few special cases of the cost model with analyses available in the literature. More detailed comparisons and a comprehensive parametric study are reported by Yao and DeJong (1978).

The access algorithm for two-variable queries in SEQUEL (Astrahan 1976) may be interpreted as follows:

> Restriction indexing on File 1 and File 2.
> For each File 1 record satisfying the restriction,
> use the join value to access the File 2 join index;
> find File 2 records satisfying both join and restriction;
> concatenate and project.

This can be described by the following query graph:



This access algorithm is similar to the type combination 11/31, except that the join index and records of File 2 are accessed repeatedly, once for each record of File 1. Since records are not sorted before joining, redundant accesses can occur if several File 1 records have the same join value. Consider the situation that for both files there exists a clustering index on the restriction attribute and a non-clustering index on the join attribute. Figure 7.1 shows that the algorithm type 31/31 has a lower cost. An interesting case is where no index structures are present. The SEQUEL algorithm calls for an index to be built on the join attribute of File 2. Restrictions are performed by se-

		clustering	non-clustering	non-existent
File 1	Restriction Index	✓		
	Join Index		✓	
File 2	Restriction Index	✓		
	Join Index		✓	

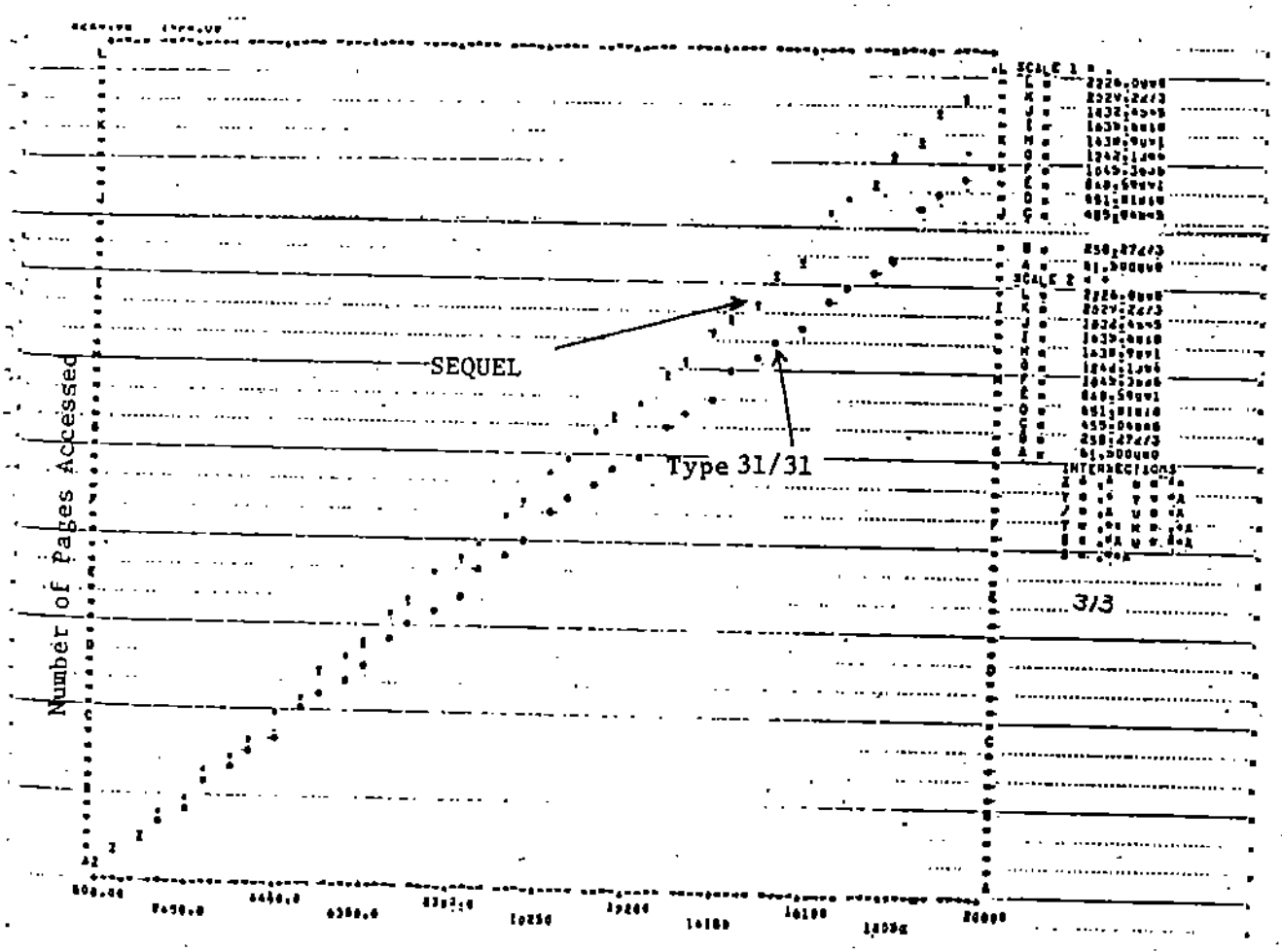


Figure 7.1 Comparison of SEQUEL and Type 31/31.

quential scans. This corresponds roughly to the type 41/21, plus building the join index. When this is compared to the type 41/41, Figure 7.2 shows that the latter has a cost reduction of almost 50% under the situation considered.

The cost of the "join indexing" algorithm in Example 1 was evaluated in Blasgen (1977). Their evaluation results can be compared with C_{21} using identical data models and storage structures. Let File 1 range in size from 500 to 20,000 records, and File 2 be always four times larger than File 1. Also assume that there exist clustering indices for both join attributes. File 1 is retrieved and restricted before File 2, allowing for "feedback" from File 1 to File 2 if any join values have been eliminated by the restriction. This feedback effect is not considered in Blasgen (1977). Figure 7.3 shows that the access cost increases as the number of records per join value increases. The reason is that as more records have a given join value, fewer join values will be eliminated by the restriction on the first file.

An approach to obtain efficient access algorithms is to develop heuristics. One such heuristic is to relocate the access operations using intuition. Smith and Chang (1975) suggested some heuristics which include 1) using indices whenever possible to reduce record accesses; 2) reducing the size of the files being accessed as soon as possible via restrictions and projections; and 3) sorting the intermediate results whenever it would improve the efficiency of the following operation. By following these rules we arrive at the algorithm type 13/13. This set of heuristics does not produce the optimal access algorithm, however, since the fact that the join filter can also reduce the file size is neglected. As an example, Figure 7.4 shows that the type 31/31 outperforms the type 13/13 for some values of the join selectivity parameter.

		clustering	non-clustering	non-existent
File 1	Restriction Index			✓
	Join Index			✓
File 2	Restriction Index			✓
	Join Index			✓

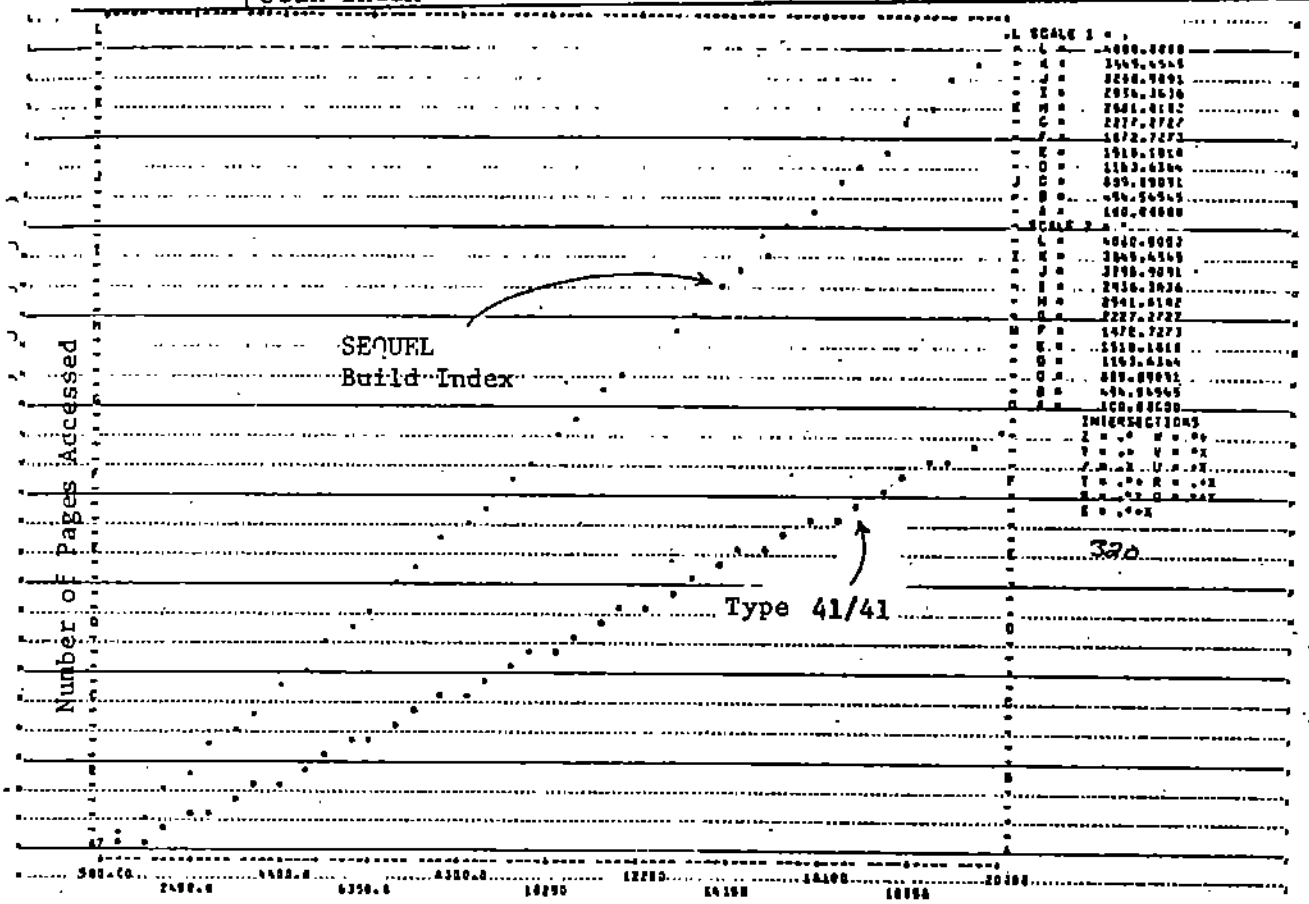
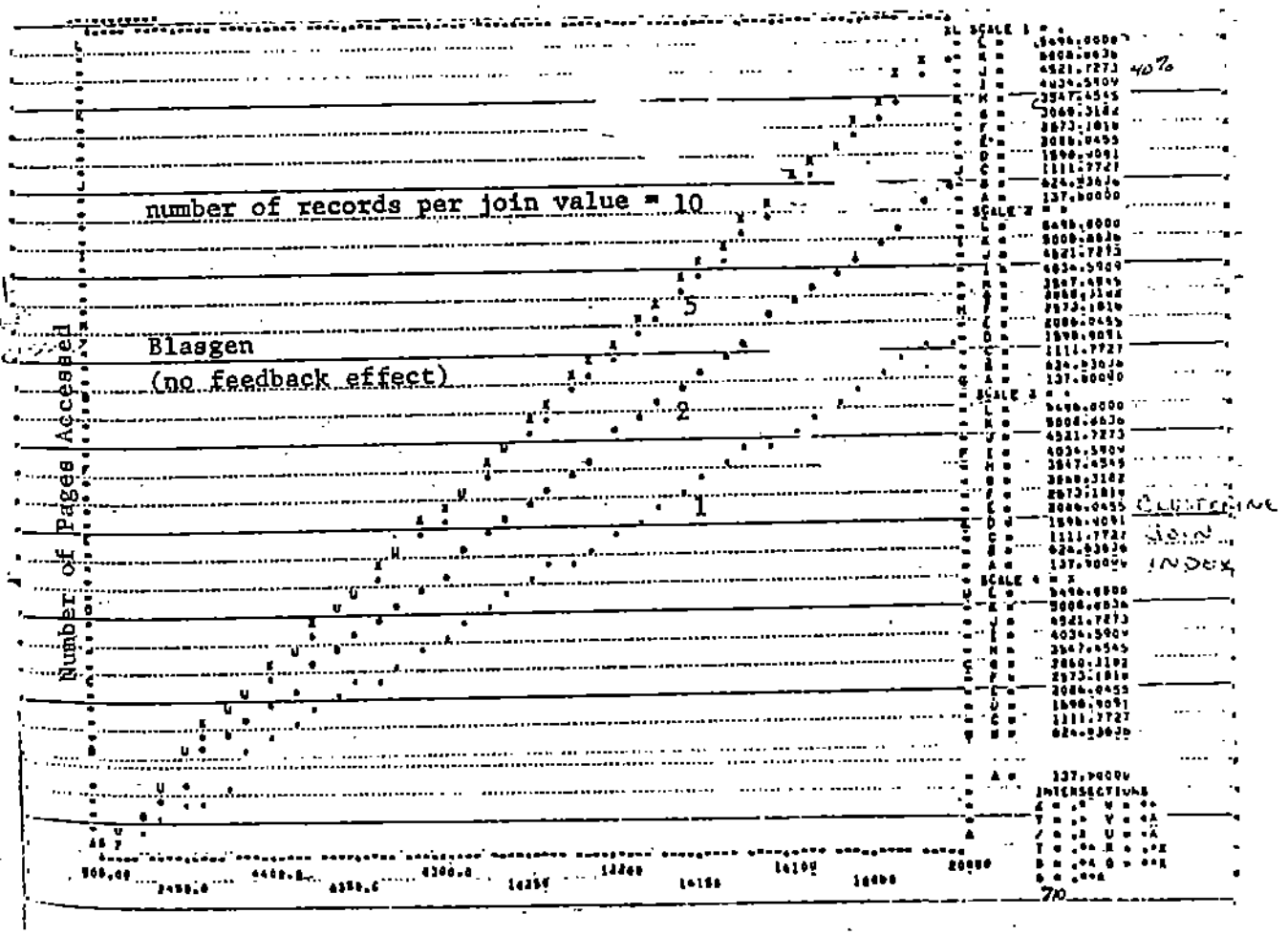
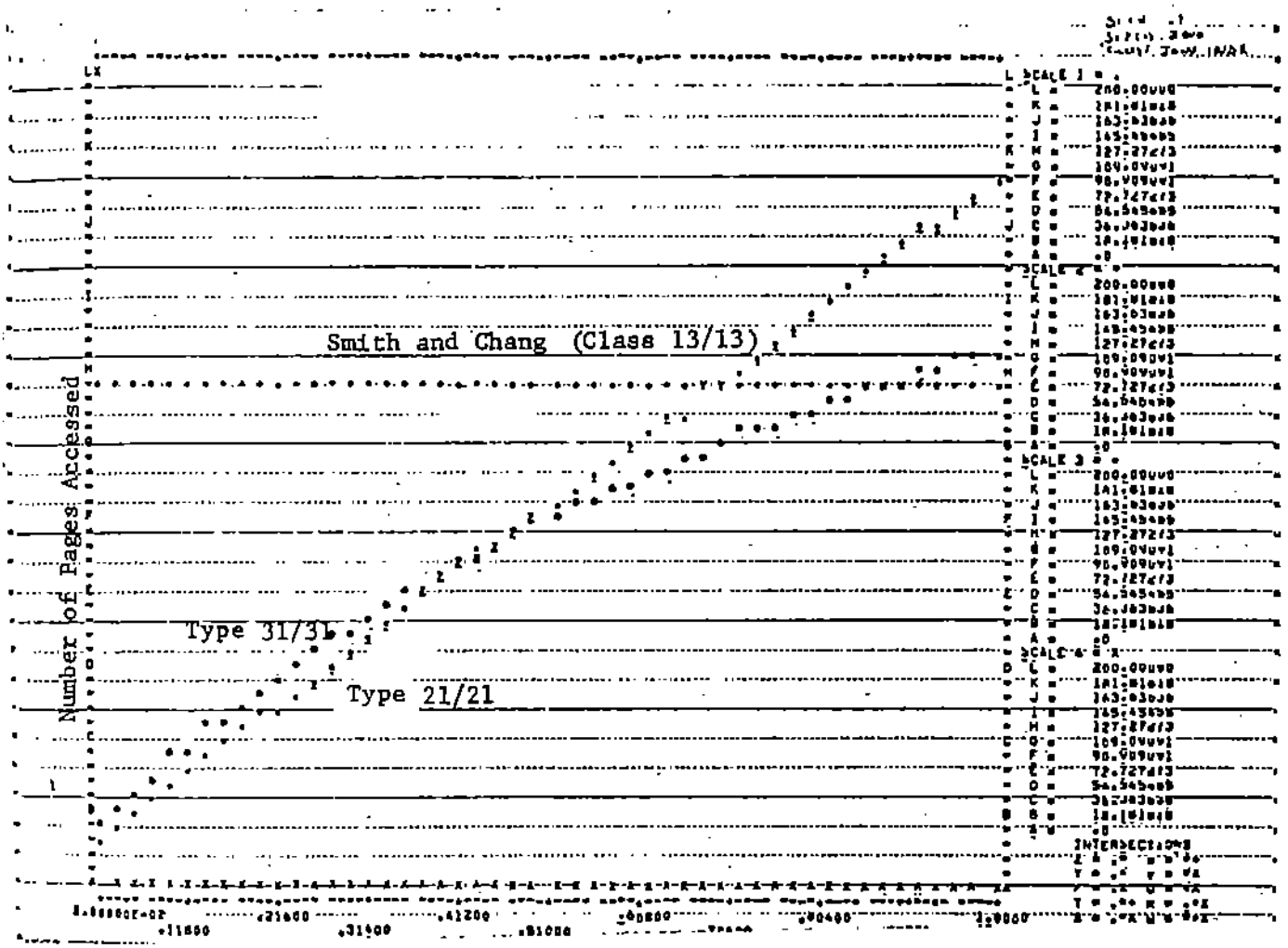


Figure 7.2 Comparison of SEQUEL and Type 41/41.



Number of Records in File 1 (File 2 size = 4 * File 1 size)

Figure 7.3 Feedback Effect as a Function of the Number of Join Values



		clustering	non-clustering	non-existent
File 1	Restriction Index		✓	
	Join Index	✓		
File 2	Restriction Index		✓	
	Join Index	✓		

2000 records in each file
Restriction selectivity = 0.1

Figure 7.4 The Effect of Join Selectivity

As there does not exist a universally optimal algorithm, many systems implemented a "compromise" algorithm (e.g. SEQUEL). Alternatively, several access algorithms may be implemented and the system will select an appropriate one every time a query is to be evaluated (Astrahan 1976). The cost model may be employed to do this selection by computing the cost equations of the implemented algorithms. In fact, the optimal access algorithm may be determined by computing all the cost equations of the cost model. That is, we first determine the best version within each class:

$$C_1 = MIN(C_{11};C_{12};C_{13})$$

$$C_2 = MIN(C_{21};C_{22};C_{23})$$

$$C_3 = MIN(C_{31};C_{32};C_{33})$$

$$C_4 = MIN(C_{41};C_{42};C_{43})$$

$$C_6 = MIN(C_{61};C_{62};C_{63};C_{64})$$

$$C_7 = MIN\ (C_{71};C_{72};C_{73};C_{74})$$

Then the optimal class is determined:

$$C = MIN(C_1;C_2;C_3;C_4;C_6;C_7)$$

The evaluation of these cost equations also takes into consideration the applicability of each class. Classes not applicable for the particular data base state are eliminated from the evaluation. Even though there are up to 20 equations to be computed for each file, they can be computed very efficiently since the equations share a large number of common terms. The optimization is essentially an exhaustive search which is shown to be effective in this case.

We note, however, that since the optimal access algorithm depends on the particular query and data base state, the optimization is not helpful unless the optimal algorithms are always included in the set of implemented algorithms. Ideally, the system should be able to dynamically construct an optimal algorithm to evaluate a given query. The query evaluation model provides a convenient basis for such an adaptive query subsystem. Since the interfaces and the pipelining degrees of the access algorithm operators are well defined, it is possible to implement these operators as a set of cooperating sequential processes. The basic query subsystem architecture is similar to that of the CONVERT run-time system (Shu et al. 1977, Smith and Chang 1975). Each operator defines a process. They communicate via a shared reentrant supervisor.

A simpler alternative is currently being implemented and tested. The access operators are implemented as subroutines of a main program which acts as the supervisor to synchronize the data and control flow. The degree of pipelining is minimized to implement only the complete-hold and the join-value-hold. The architecture of this simplified system is shown in Figure 7.5 . The query analyzer decomposes a general query into two variable queries. The optimizer selects an optimal access algorithm class. The execution monitor is a program which "synthesizes" the optimal algorithm by calling the appropriate access operator routines. The detailed structure, implementation and performance of such a self-optimizing query subsystem will be reported in a forthcoming paper (Yao 1978b).

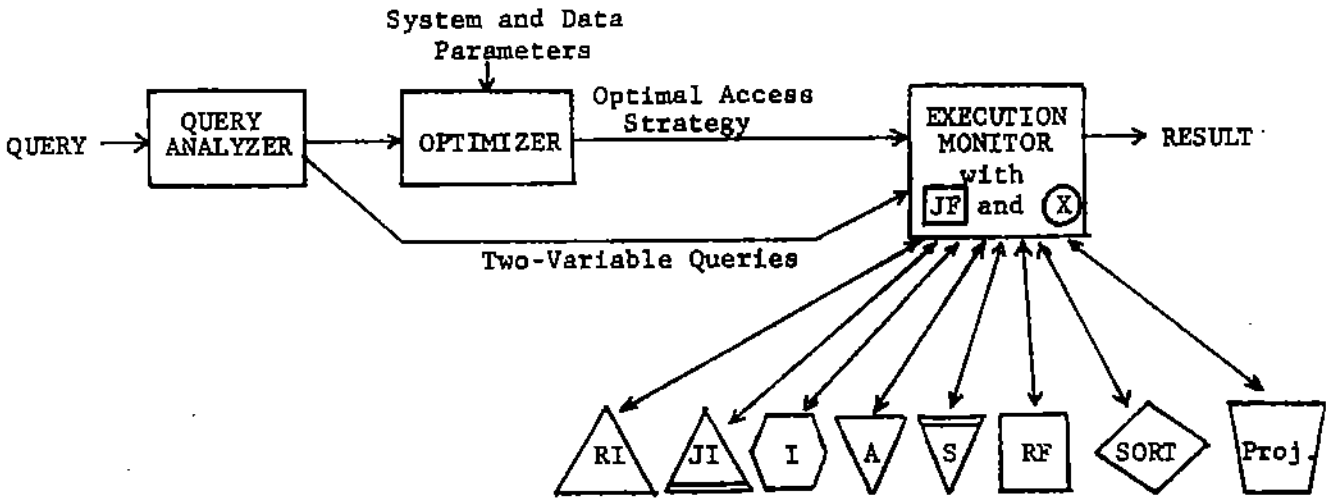


Figure 7.5 Query Evaluation Subsystem

8. Conclusions

A model of data base query evaluation is presented. Using the model, the access cost of query is analyzed and optimized. The advantage of this approach is that it helps to decomposea complex access strategy into simple operations. Unlike the heuristics and the automatic programming approaches, the optimization of query evaluation using the model can be implemented effectively and efficiently. The optimizer takes into account the storage structures of a data base system and minimizes the response time in terms of secondary storage accesses. The adaptive query subsystem can be constructed with a modular architecture using components found in conventional query subsystems.

It is observed that although the present model and system are developed in the context of relational systems, it can be applied to systems that support other data models. The model also enables the consideration of parallelism and pipelining concepts whose potential can only be revealed in parallel processing and dedicated data base computers.

REFERENCES

Astrahan, M.M. and D.D. Chamberlin, "Implementation of a structured English query language," Comm. ACM, Vol. 18, No. 10 (Oct. 1975), pp. 580-588.

Astrahan, M.M. et al., "System R: A relational approach to data base management," ACM Transactions on Data Base Systems, Vol. 1, No. 2 (June 1976), pp. 97-137.

Bernstein, P. et al., "A framework for query optimization" (in preparation).

Blasgen, M.W. and K.P. Eswaran, "Storage access in relational data bases," IBM Systems J., No. 4, 1977, pp. 363-377.

Codd, E.F., "A relational model of data for large shared data banks," Comm. ACM, Vol. 13, No. 6 (June 1970), pp. 377-387.

Codd, E.F., "A data base sublanguage founded on the relational calculus," Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access and Control, New York: ACM.

Gotleib, L., "Computing joins of relations," Proc. ACM-SIGMOD Conf., San Jose, California, May 1975, New York: ACM, pp. 53-63.

Information Management System, General Information Manual, Form GH20-0765, IBM Corp., Data Processing Division, White Plains, New York 10604.

Knuth, D., The Art of Computer Programming, Vol. 3, Reading, Massachusetts: Addison-Wesley, 1973.

Palermo, E.P., "A data base search problem," Proc. 4th Int'l Symp. on Computers and Information Science, Miami Beach, Fla., Dec. 1972.

Pecherer, R.M., "Efficient evaluation of expressions in a relational algebra," Proc. ACM Pacific 75 Conf., April 1975, pp. 44-49.

Rothnie, J.B., "Evaluating inter-entry retrieval expressions in a relational data base management system," Proc. AFIPS 1975 NCC, Vol. 44, Montvale, New Jersey: AFIPS Press, pp. 417-423.

Schkolnick, M., "A clustering algorithm for hierarchical structures," ACM Transactions on Data Base Systems, Vol. 2, No. 1 (March 1977), pp. 27-44.

Shu, N.C., B.C. Housel, R.W. Taylor, S.P. Ghosh, and V.Y. Lum, "EXPRESS: A Data EXtraction, Processing and REStructuring System," ACM Transactions on Database Systems, Vol. 2, No. 2, June 1977.

Smith, J.M. and P.Y.T. Chang, "Optimizing the performance of a relational algebra database interface," Comm. ACM, Vol. 18, No. 10 (October 1975), pp. 568-579.

Smith, J.M., and P.Y.T. Chang, "Optimizing the performance of a relational algebra database interface," Comm. ACM, Vol. 18, No. 10 (October 1975), pp. 568-579.

Wong, E. and K. Youssefi, "Decomposition--a strategy for query processing," ACM Transactions on Data Base Systems, Vol. 1, No. 3 (September 1976), pp. 223-241.

Yao, S.B., "An attribute based model for database access cost analysis," ACM Transactions on Database Systems, Vol. 2, No. 1 (March 1977), pp. 45-67.

Yao, S.B., "Approximating block accesses in data base organizations," Comm. ACM, Vol. 20, No. 4 (April 1977), pp. 260-261.

Yao, S.B., "User criteria for query languages," in preparation, 1978a.

Yao, S.B., "A self-optimizing query subsystem," in preparation, 1978b.

Yao, S.B. and D. DeJong, "Evaluation of database access paths," unpublished manuscript, 1978.

APPENDIX A

The index selection algorithm developed by Astrahan (1976) is extended to the case of a Conjuctive Normal Form predicate $D = (c_{11} v \ldots v c_{1n_1}) \wedge \ldots \wedge (c_{m1} v \ldots v c_{mn_m})$. A tree representation of the predicate D is shown in Figure A.1. Each c_{ij} in D is a clause whose attribute(s) may be indexed. Let A_{ij} contain the attribute(s) of c_{ij}.

The index selection is based on the observation that indices in a disjunction $(c_{i1} v \ldots v c_{in_i})$ do not reduce the range of file searching unless all attributes in the disjunction are indexed while any index in a conjunct helps to reduce the search range. Using the conjunctive predicate tree, we label each OR node by the algorithm:

If all attributes in $\bigcup_{j=1}^{n_i} A_{ij}$ are indexed,

then $\text{LABEL}(\text{OR}_i) = \bigcup_{j=1}^{n_i} A_{ij}$

else $\text{LABEL}(\text{OR}_i) = \emptyset$.

We then label the AND node as $\text{LABEL}(\text{AND}) = \bigcup_{i=1}^{m} \text{LABEL}(\text{OR}_i)$. The set LABEL(AND) contains the attributes whose indices are selected for accessing.

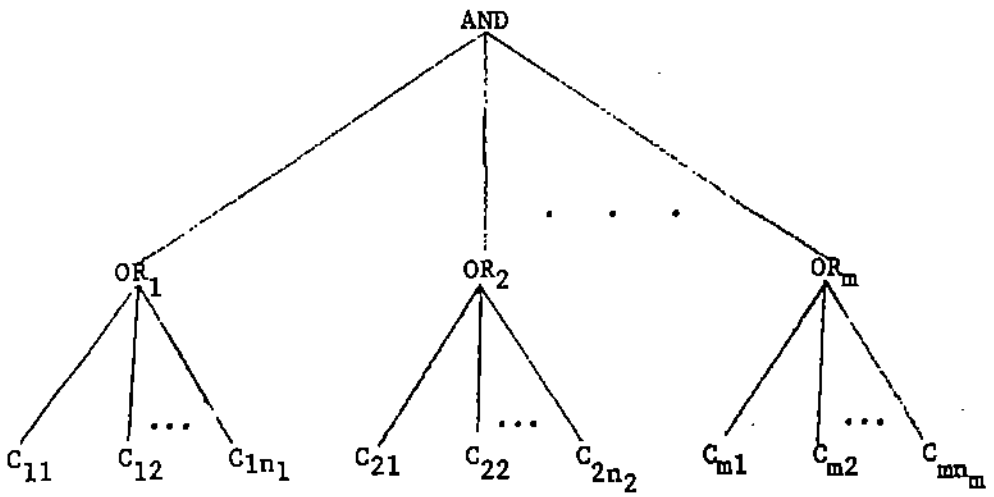


Figure A.1 A Conjunctive Predicate Tree

APPENDIX B

The cost of access operations is derived in terms of page accesses. The definition of parameters used can be found in Figure 6.1.

A. Restriction Indexing

An index may have to be accessed for more than one value, such as in the case of the disjunction $(a = v_1) \vee (a = v_2)$ where two values of the attribute a need to be accessed. Denote u_{jk} as the number of values accessed for the index of d_{jk}. The restriction indexing cost is the sum of the costs for each index access.

In searching a value in an index, one page for each level of the index must be accessed. If the size of the file F_i is r_i and the degree of the index is z, this process requires $\ell = \log_z r_i$ accesses. The cost for searching the index of d_{jk} is $u_{jk}\log_z r_i$, and the cost for searching all indices is $\sum_{u=1}^{m}\sum_{k=1}^{n_j} u_{jk}\log_z r_i$. Restriction indexing produces, from the index of d_{jk}, a set of pointers to the records satisfying the corresponding clauses. Assuming these pointers are sequentially stored in the indices, the number of pages containing the pointers in the index of d_{jk} is $(u_{jk}r_i s_{jk})/b$ where s_{jk} is the selectivity of the index of d_{jk} and b is the page size. The number of pointer pages needed to be accessed for all indices is $\sum_{j=1}^{m}\sum_{k=1}^{n_j}(u_{jk}\cdot r_i \cdot s_{jk})/b$. Assuming these pointers are combined into one pointer set at no cost, the average total cost for restriction indexing is given by

$$R(r_i) = \sum_{j=1}^{m}\sum_{k=1}^{n_j} u_{jk}\left(\log_z r_i + \frac{r_i s_{jk}}{b}\right).$$

B. Join Indexing

In this case, the index is used to obtain pointers to the entire file. Assuming it is possible to search the index sequentially, the number of page accesses is simply equal to the number of pages containing the record pointers. There are r_i pointers since the index is dense. The average access cost for join indexing is $I(r_i) = \frac{r_i}{b}$.

C. Record Access

The input to this component is a set of pointers to records in the file F_i. The pointers are assumed to be ordered in increasing record addresses. Since the file may be clustered with respect to one of the indices that produce the pointer set, there are two parameters to be considered:

$$\alpha = \begin{cases} \text{the selectivity of the clustering index} \\ 0 \text{ if there is no clustering index} \end{cases}$$

$$\beta = \begin{cases} \text{the product of the nonclustering index selectivities} \\ 0 \text{ if there is no nonclustering index} \end{cases}$$

For the non-clustering indices, the file organization is independent of the index structures. It is assumed that the records corresponding to the input pointers are distributed randomly over the file. If the file F_i has r_i records , then the total number of records accessed by non-clustering indices is $y = r_i \beta$. This corresponds to approximately $x = p_i(1-(1-1/p_i)^y)$ randomly accessed pages.

The consecutive pointers of the clustering index will point to consecutive records in the file. However, since the file F_i may be clustered with other files, the consecutive records in the file F_i may not be physically consecutive in storage. There are αr_i consecutive pointers in the clustering index. Let C_i indicate the children files of the file i. The number of pages accessed for the clustering index may be estimated by $\alpha r_i P_i$ where $P_i = MIN(1; d_i/b)$ and $d_i = f_i + \sum_{j \in C_i} k_{ij} d_j c_{ij}$ are the "twin distance equations" of Schkolnick (1977). The probability for a page to

be accessed by the clustering index can be computed by $(\alpha r_i P_i)/p_i$. Therefore, the combined number of page accesses by the predicate is estimated by $A(\alpha,\beta) = x + (p_i - x)\frac{(\alpha r_i P_i)}{p_i}$.

We note that when there is no non-clustering index ($\beta=0$), the equation is reduced to $A(\alpha,0) = \alpha r_i P_i$ and when there is no clustering index ($\alpha = 0$), the equation is reduced to $A(0,\beta) = p_i(1-(1-1/p_i)^{P_i})$.

D. Sequential Scan

If there exists no access path for retrieving records in the file F_i, the file may be accessed by sequentially scanning the storage area in which the file is stored. If the storage area has e_i pages, the access cost for the sequential scan is simply $S(e_i) = e_i$.

E. Sorting

Sorting records on the join attribute represents an important cost factor in the access algorithms. If there are r records (each of size f_i) to be sorted, then it requires $\frac{r \cdot f_i}{b}$ pages to store these records. Assuming the I/O transfers one page at a time, it is well known that an n-way sort-merge requires $T(r,f_i) = 2\frac{r \cdot f_i}{b} \log_n \frac{r \cdot f_i}{b}$ accesses (Blasgen, Knuth).

F. Join and Concatenation

butes and produce record concatenations as output. For r records (each of size f_i), the join cost is estimated by the cost of accessing the records in one pass.

$$J(r,f_i) = \frac{r \cdot f_i}{b}.$$

It is assumed that the cost of concatenation and preparation of output is estimated also by

$$C(r,f_i) = \frac{r \cdot f_i}{b}.$$

Concatenation may sometimes be performed on pointers to produce pairs of pointers for records satisfying the join. For each join value, there will be

rs_i pointers, and hence approximately $(rs_i)^2$ pointer pairs. There are $\frac{1}{s_i}$ join values and the join selectivity t_i limited this to $\frac{1}{s_i}\ t_i$ values. Therefore the total number of pointer pairs is approximately $(rs_i)^2\frac{1}{s_i}\ t_i = r^2 s_i t_i$. It is assumed that the concatenation cost is estimated by

$$C_1(r) = \frac{r^2 s_i t_i}{b} .$$

G. Projection

The projection operation removes the unwanted items from records in a file and the duplicated records thus created. The latter task has a dominant cost. In order to discover duplicates, the file must be sorted completely on all attributes. In practice we assume that sorting on two attributes is sufficient. If the file containing r records (each of size f_i) is previously sorted on the join attribute, then it requires one additional sort. Since the join attribute a_i partitions the file into $1/s_i$ subfiles containing the same join values, the cost for this sorting is

$$P(r) = \frac{1}{s_i} T(rs_i, f_i).$$

If the file is not previously sorted, then there is an added cost $T(r,f_i)$ to sort the file. However, if the projection attributes include the primary key, one sort on the primary key is sufficient. Let $\delta=0$ indicate that the primary key is included in the projection; $\delta=1$ otherwise. The projection cost for an unsorted file is given by

$$P_\delta(r) = T(r,f_i) + \delta\frac{1}{s_i}\ T(rs_i, f_i).$$

Projection reduces both the record size and the file size. Let g_i be the reduced record size of file F_i after projection. The projected file may have a maximum size defined by the Cartesian product of the projected values: $h = \prod_{a_j \in A} \frac{1}{s_j}$. It is estimated that projection reduces the file size from r to

$$H(r) = MIN(r;h).$$

H. Link Access (1:m)

The (1:m) link access is commonly referred to as parent-child searching. Let t_j be the join selectivity for the file F_j. After the parent file F_j has been restricted and accessed, there will be $r_j t_j$ link pointers to be traced to the child file F_i. Consider the following two cases:

i) Parent and child files not clustered (c_{ij}=0): The number of child records for each parent record can be estimated by $r_i t_i$ where r_i is the child file size and t_i is the selectivity of the join attribute defining the link. There are two sub-cases:

a) If the child file is not clustered with respect to the join attribute, then one access is required for each child record.

b) If the file is clustered with respect to the join attribute, then it requires one access for the first child, and the twin accesses are computed by the distance equation. The cost for each link is $1 + (r_i t_i - 1)P_i$.

Summing up the two sub-cases, we have the access cost $r_j t_j(c_i(1+(r_i t_i-1)P_i)+(1-c_i)r_i t_i)$ where c_i=1 indicates that the child file is clustered with respect to the join attribute.

ii) Parent and child clustered (c_{ij}=1): The number of page accesses required for the first child is computed by the "parent-child distance equation" in (Schkolnick 1977): P_{ji}=MIN(1;d_{ji}) where $d_{ji} = f_j + \sum_{\substack{\ell \in C_j \\ \ell<i}} k_{j\ell} d_\ell c_{j\ell}$. The

twins' access cost is $(r_i t_i - 1)P_i$. The cost in this case is $r_j t_j(P_{ji}+(r_i t_i - 1)P_i)$.

Summing up all cases, the link access cost is

$$c_{ij} r_j t_j (P_{ji}+(r_i t_i - 1)P_i)+(1-c_{ij}) r_j t_j (c_i(1+(r_i t_i - 1)P_i)+(1-c_i) r_i t_i).$$

The cost equation may be rewritten in the general form:

$$L(\alpha',\beta') = c_{ij}\beta'(P_{ji}+(\alpha'-1)P_i)+(1-c_{ij})\beta'(c_i(1+(\alpha'-1)P_i)+(1-c_i)\alpha')$$

where α is the number of parents and β is the number of children.

I. Link Access (m:1)

The (m:1) link access is commonly referred to as child-parent searching. After the child file F_j is accessed, the parent records in the parent file F_i are accessed. On the average, every group of $r_j t_j$ records in the child file shares a same parent record in the file F_i. With proper implementation, it is possible to access the parent record only once for all the $r_j t_j$ children records. We assume that the parent record is accessed from the first child record and saved for other children records. Consider the following two cases:

i) Parent and child files not clustered ($c_{ij}=0$): If the join selectivity on the file F_i is t_i, then there are $r_i t_i$ parent records to be accessed. One access is required for each parent record, and the total cost is $r_i t_i$.

ii) Parent and child files clustered ($c_{ij}=1$): Since it is assumed that the parent is accessed from the first child, the cost for accessing one parent is the same as the parent-child access cost P_{ij}. The total cost is $r_i t_i P_{ij}$.

Summing up the two cases, the (m:1) link access cost is given by:

$$c_{ij} r_i t_i P_{ij}+(1-c_{ij}) r_i t_i$$

and the general form is

$$L'(\alpha') = c_{ij}\alpha' P_{ij}+(1-c_{ij})\alpha'.$$

APPENDIX C

After searching the restriction indices, records satisfying the access predicate $(d_{11}v\ldots vd_{1n_1})\wedge\ldots\wedge(d_{m1}v\ldots vd_{mn_m})$ are accessed. Consider the j'th disjunction $(d_{j1}v\ldots vd_{jn_j})$. The weighted selectivity of d_{jk} is $u_{jk}s_{jk}$ where u_{jk} is the number of values specified by d_{jk} and s_{jk} is the selectivity of the index of d_{jk}. The selectivity (probability) for a record to be accessed by the j'th disjunction is

$$q_j = \sum_{k=1}^{n_j} u_{jk}s_{jk}\prod_{w=1}^{k-1}(1 - u_{jw}s_{jw})$$

$$= u_{j1}s_{j1}+(1-u_{j1}s_{j1})u_{j2}s_{j2}+\ldots+(1-u_{j1}s_{j1})(1-u_{j2}s_{j2})\ldots(1-u_{j(n_j-1)}s_{j(n_j-1)})u_{jn_j}s_{jn_j}$$

and the selectivity of the predicate is $q = q_1q_2\ldots q_m$.

It is possible that one of the indices is a clustering index. Consider the j'th disjunction $(d_{j1}v\ldots vd_{jn_j})$ in which the first clause d_{j1} has a clustering index. The weighted selectivity of the clustering index is $q_j' = u_{j1}s_{j1}$. The selectivity of non-clustering indices in the j'th disjunction is

$$q_j'' = \sum_{k=2}^{n_j} u_{jk}s_{jk}\prod_{w=1}^{k-1}(1-u_{jw}s_{jw}).$$

Since all other disjunctions contain only non-clustering indices, the selectivity of the non-clustering indices is $q_1\ldots q_{j-1}q_j''q_{j+1}\ldots q_m$.

We summarize the above derivation by defining the clustering index selectivity γ and the non-clustering indices selectivity ξ as:

$$\gamma = \begin{cases} q_1\ldots q_{j-1}q_j'q_{j+1}\ldots q_m & \text{if there is a clustering index for some } d_{j1} \\ 0 & \text{otherwise.} \end{cases}$$

$$\xi = \begin{cases} q_1\ldots q_{j-1}q_j''q_{j+1}\ldots q_m & \text{if there is a clustering index for some } d_j \\ q_1\ldots q_m & \text{otherwise.} \end{cases}$$

$$q = \gamma + \xi$$

Recall that the access predicate is a sub-predicate of the original predicate $C = (d_{11} v \ldots v d_{1a_1}) \wedge \ldots \wedge (d_{b1} v \ldots v d_{ba_b})$. The selectivity of C is more "restrictive" than that of the access predicate, and can be similarly defined as follows:

$$q = \hat{q}_1 \hat{q}_2 \ldots \hat{q}_b$$

where

$$\hat{q}_j = \sum_{k=1}^{a_j} u_{jk} s_{jk} \prod_{w=1}^{k-1} (1 - u_{jw} s_{jw}).$$